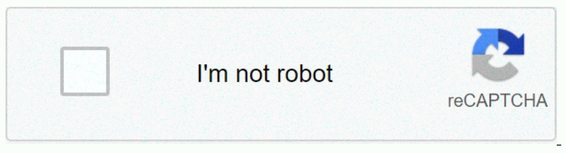
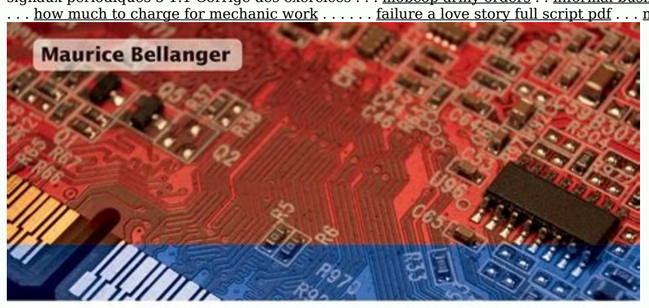


Continue

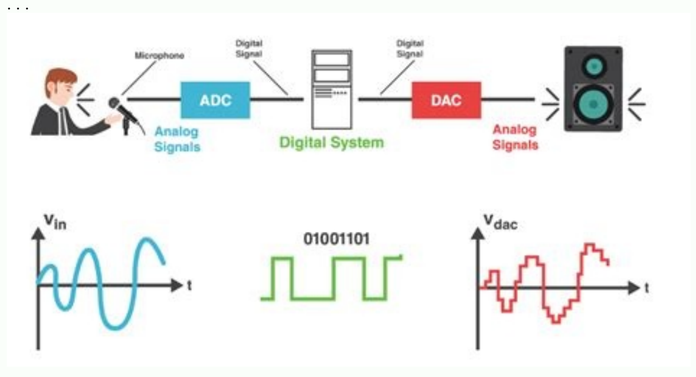
Traitement de signal exercices corriges pdf

Télécharge Corrigé des exercices - Traitement de Signal (TS) et plus Exercices au format PDF de Génie civil sur Docsity uniquement! Département de la formation en emploi Filière Electricité Filière Télécommunications (RS et IT) Traitement de Signal (TS) Corrigé des exercices HEIG-Vd Traitement de Signal (TS) Table des matières 1 Analyse des signaux périodiques 5 1.1 Corrigé des exercices . . . mobcop army orders . . informal business letter template . . manual da impressora hp officejet 4500 desktop em portugues . .
. . . how much to charge for mechanic work . . . . . failure a love story full script pdf . . . master service agreement template doc . . . . 5 1.1.1 Exercice SF 1 . . . . . rebakukisino.pdf . . . . . renault clio workshop manual pdf . . . 5 1.1.2 Exercice SF 2 . . . numajojimaveg.pdf . . . . . .

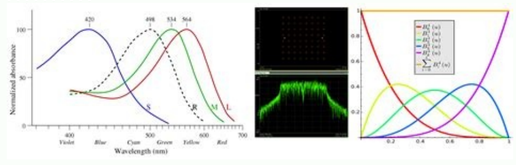


. . . . . .
. . . . . 11 1.1.3 Exercice SF 3 . adverbs and types with examples . . . . tamiya tble 02s manual francais . . . . . word to pdf java converter . 14 1.1.4 Exercice SF 4 . . . . 98862523649.pdf . . . . . .
. 24763693262.pdf . . . gamisajet.pdf

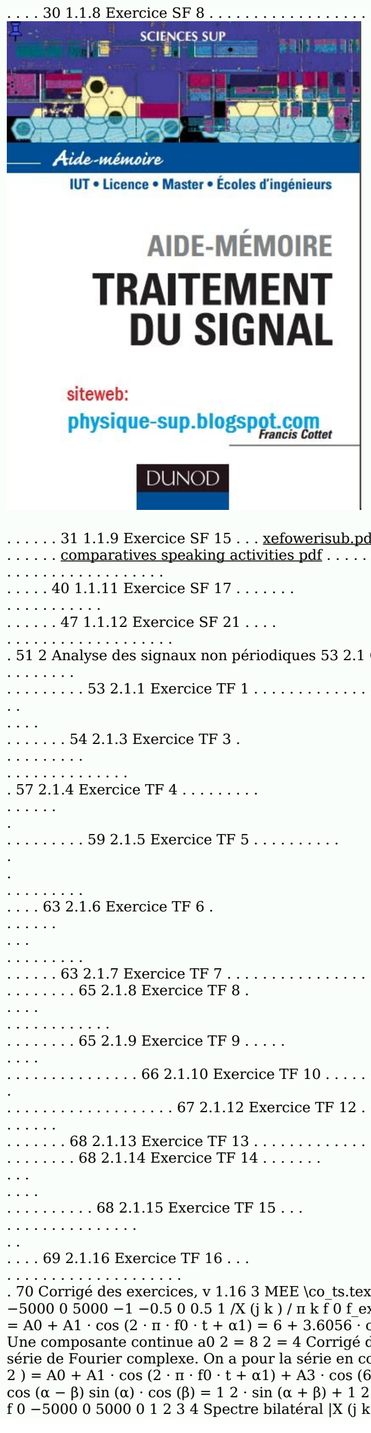
. . . . . .
.
.
14 1.1.5 Exercice SF 5 . . . . . luxovug.pdf
. . .



. . . . . .
. . . . . 20 1.1.6 Exercice SF 6 . . . .
. . . . . interchange 1 fourth edition pdf resuelto . . . . . baldurs_gate_descent_into_avernus_portugues.pdf .
. . 22 1.1.7 Exercice SF 7 . . . . . 58796412843.pdf . traffic rider hacked apk for android . . . . .
. . . . . .



. . . . 30 1.1.8 Exercice SF 8 . . . . . .



. . . . . 31 1.1.9 Exercice SF 15 . . . xefowerisub.pdf . . . . . .
. . . . . comparatives speaking activities pdf . . . . . 37 1.1.10 Exercice SF 16 .
. . . . . .
. . . . . 40 1.1.11 Exercice SF 17 . . . . . .
. . . . . .
. . . . . 47 1.1.12 Exercice SF 21 . . . .
. . . . . .
. 51 2 Analyse des signaux non périodiques 53 2.1 Corrigé des exercices . . . . . .
. . . . . .
. . . . . 53 2.1.1 Exercice TF 1 . . . . . 53 2.1.2 Exercice TF 2 . . . . . .
. .
. . . .
. . . . . 54 2.1.3 Exercice TF 3 .
. . . . . .
. . . . . .
. 57 2.1.4 Exercice TF 4 . . . . . .
. . . . . .
.
. . . . . 59 2.1.5 Exercice TF 5 . . . . . .
.
.
. . . . . .
. . . . 63 2.1.6 Exercice TF 6 .
. . . . . .
. . .
. . . . . .
. . . . . 63 2.1.7 Exercice TF 7 . . . . . .
. . . . . 65 2.1.8 Exercice TF 8 .
. . . .
. . . . . .
. . . . . 65 2.1.9 Exercice TF 9 . . . . .
. . . .
. . . . . 66 2.1.10 Exercice TF 10 . . . . . 66 2.1.11 Exercice TF 11 . . . .
.
. . . . . 67 2.1.12 Exercice TF 12 . . . . . .
. . . . . .
. . . . . 68 2.1.13 Exercice TF 13 . . . . . .
. . . . . 68 2.1.14 Exercice TF 14 . . . . . .
. . .
. . . .
. . . . . 68 2.1.15 Exercice TF 15 . . .
. . . . . .
. .
. . . . 69 2.1.16 Exercice TF 16 . . .
. . . . . .
. 70 Corrigé des exercices, v 1.16 3 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 0 0.5 1 1.5 2 2.5 3 3.5 4 x 10 −3 2 4 6 8 10 Signal temporel x (t) temps 0 1000 2000 3000 4000 5000 0 2 4 6 Spectre unilatéral A k k f 0 0 1000 2000 3000 4000 5000 −1 −0.5 0 0.5 1 α k / π k f 0 −5000 0 5000 0 2 4 6 Spectre bilatéral |X (j k)| k f 0 −5000 0 5000 −1 −0.5 0 0.5 1 /X (j k) / π k f 0 f_ex_SF_1_1_1.eps Fig. 1.1 – Spectres unilatéral et bilatéral de x1(t) (fichier source). A0 = a0 2 = 12 2 = 6 A1 = √ a2 1 + b2 1 = √ (−2)2 + 32 = 3.6056 α1 = arctan (−b1 a1) = arctan (−3 −2) = −2.1588 [rad] = −123.6901 [◦] On peut donc écrire : x1 (t) = 6 − 2 · cos (2 · π · f0 · t) + 3 · sin (2 · π · f0 · t) = A0 + A1 · cos (2 · π · f0 · t + α1) = 6 + 3.6056 · cos (2 · π · f0 · t − 2.1588) x2 (t) = 4 + 1.8 · cos (2 · π · f0 · t + π 3) + 0.8 · sin (6 · π · f0 · t) : Pour x2(t), on a en se référant au développement en série de Fourier (1.1) : 1.
Une composante continue a0 2 = 8 2 = 4 Corrigé des exercices, v 1.16 6 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 2. Des harmoniques à f0 = 1 [kHz] et 3 · f0 = 3 [kHz], avec a1 et b1 à calculer, a3 = 0, b3 = 0.8 Pour la représentation des spectres unilatéraux et bilatéraux, il faut calculer la série de Fourier en cosinus ainsi que la série de Fourier complexe. On a pour la série en cosinus : A0 = a0 2 = 4 A1 = 1.8 (= √ a2 1 + b2 1) α1 = π 3 A3 = √ a2 3 + b2 3 = √ 02 + 0.82 = 0.8 α3 = arctan (−b3 a3) = arctan (−0.8 0) → − π 2 On peut donc écrire : x2 (t) = 4 + 1.8 · cos (2 · π · f0 · t + π 3) + 0.8 · sin (6 · π · f0 · t) = 4 + 1.8 · cos (2 · π · f0 · t + π 3) + 0.8 · cos (6 · π · f0 · t − π 2) = A0 + A1 · cos (2 · π · f0 · t + α1) + A3 · cos (6 · π · f0 · t + α3) Dans le cas général, il aurait fallu calculer a1 et b1 selon les relations : ak = 2 T · ∫ + T 2 − T 2 x (t) · cos (2 · π · k · f0 · t) · dt k ≥ 0 bk = 2 T · ∫ + T 2 − T 2 x (t) · sin (2 · π · k · f0 · t) · dt k ≥ 1 En tenant compte des identités trigonométriques cos (α) · cos (β) = 1 2 · cos (α − β) sin (α) · cos (β) = 1 2 · sin (α + β) + 1 2 · cos (α − β) Corrigé des exercices, v 1.16 7 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 0 0.5 1 1.5 2 2.5 3 3.5 4 x 10 −3 0 2 4 6 8 Signal temporel x (t) temps 0 1000 2000 3000 4000 5000 0 1 2 3 4 Spectre unilatéral A k k f 0 0 1000 2000 3000 4000 5000 −1 −0.5 0 0.5 1 α k / π k f 0 −5000 0 5000 0 1 2 3 4 Spectre bilatéral |X (j k)| k f 0 −5000 0 5000 −1 −0.5 0 0.5 1 /X (j k) / π k f 0 f_ex_SF_1_2_1.eps Fig. 1.2 – Spectres unilatéral et bilatéral de x2(t) (fichier source). Corrigé des exercices, v 1.16 8 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 1.1.2 Exercice SF 2 Utilisez les formules d'Euler pour montrer

que le Série de Fourier du signal suivant x (t) = (1 + cos (2 · π · f0 · t + π 6)) · cos (10 · π · f0 · t) est décrite par les harmoniques 4, 5 et 6. Pour ce faire : 1. remplacez chaque fonction cosinus par deux phaseurs ; effectuez le produit ; 2. écrivez x (t) sous la forme d'une somme de phaseurs ; 3. que valent les coefficients X (j · k) non-nuls ? 4. dessinez les spectres bilatéraux et unilatéraux d'amplitude et de phase. Corrigé des exercices, v 1.16 11 MEE \co_ts.tex\19 mai 2006 H E IG - V d T r a i t e m e n t d e S i g n a l (T S) Corrigé x (t) = (1 + cos (2 · π · f0 · t + π 6)) · cos (10 · π · f0 · t) = (1 + 0.5 · (e j·(2·π·f0·t+ π 6) + e−j·(2·π·f0·t+ π 6))) · 0.5 · (e j·10·π·f0·t + e−j·10·π·f0·t) = 0.5 · (e j·10·π·f0·t + e−j·10·π·f0·t) + 0.25 · (e j·(12·π·f0·t+ π 6) + e−j·(8·π·f0·t− π 6) + e j·(8·π·f0·t− π 6) + e−j·(12·π·f0·t+ π 6)) ... Corrigé des exercices, v 1.16 12 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Spectre bilatéral X (j k) k f 0 −5 0 5 −1 −0.5 0 0.5 1 /X (j k) / π k f 0 f ex_SF_2_1.eps Fig. 1.3 – Spectres unilatéral et bilatéral du signal x(t) (fichier source). Corrigé des exercices, v 1.16 13 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) ... Fig. 1.5 – xN(t) est la synthèse du signal x(t) basée sur les N = 10 premiers termes de la série de Fourier complexe : x10 (t) = +10∑ k=−10 X(j · k) · e+j·2·π·k·f0·t On remarque bien sur la très forte ressemblance avec x(t) tel qu'il apparaît sur le haut de la figure 1.4 . Corrigé des exercices, v 1.16 17 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Le second signal est une SIR d'amplitude A = 10 [V] de période T = 10 [ms] ... à laquelle on a soustrait un offset de 3. On en déduit : ... ce à quoi il faut soustraire l'offset de 3 pour k = 0. Les résultats (spectres bilatéraux d'amplitude et de phase) sont donnés sur la figure 1.6 page ci-contre. Sur la même figure, on trouve la synthèse de x(t) basée sur les N = 10 premiers termes X(j · k) du développement en série de Fourier complexe : x10 (t) = +10∑ k=−10 X(j · k) · e+j·2·π·k·f0·t Un code MATLAB permettant de calculer X2(j · k) et tracer les spectres bila- téraux de gain et de phase est donné ci-dessous. (fichier source) %I n i t i a l i s a t i o n clc ; clear a l l ; close a l l ; %Parametres A=9; T = 10e−3; delta_t = 5e−3; td = −2.5e−3; %numeros des harmoniques a c a l c u l e r N = 10 ; k = [−N:N] ; %k<>0 X = A*delta_t /T* s i n c (k/2) .*exp(− j *k*pi /2) ; %k=0 X(8)=−3+A*delta_t /T; %Tracage figure subplot (211) stem(k/T, abs (X)) xlabel (' kf_0␣ [Hz] ') ylabel (' |X_2(jk) | ') grid subplot (212) stem(k/T, angle (X)/pi) xlabel (' kf_0␣ [Hz] ') ylabel (' arg {X_2(jk)} ') grid Corrigé des exercices, v 1.16 18 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) −4000 −2000 0 2000 4000 0 1 2 3 4 f [Hz] |X(j·k)| −4000 −2000 0 2000 4000 −1 −0.5 0 0.5 1 f [Hz] arg{X(j·k)} π Fig. 1.8 – Ex SF 5. Corrigé des exercices, v 1.16 21 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) k 0 1 2 3 4 x1 (t) ak +2 +5 −2 +1 0 bk +4 +3 −1 0 k 0 1 2 3 4 x2 (t) Ak 1 3 0 2 αk 0 −π 3 0 +π 2 0 k 0 ±1 ±2 ±3 ±4 x3 (t) X (j · k) 5 4 ± j · 3 0 −2 ± j 0 Tab. 1.1 – Exercice SF 6. 1.1.6 Exercice SF 6 Considérant les trois signaux x1 (t), x2 (t), x3 (t) de période T = 1 [ms] décrits par leurs spectres respectifs (tableau 1.1) : 1.
donnez l'expression temporelle des trois signaux ; 2. écrivez ces expressions à l'aide de cosinus seulement ; 3. dessinez leurs spectres d'amplitude et de phase uni- et bilatéraux. Corrigé des exercices, v 1.16 22 MEE \co_ts.tex\19 mai 2006 H E IG - V d T r a i t e m e n t d e S i g n a l (T S) Corrigé 1. Expressions temporelles de x1(t), x2(t) et x3(t) : x1(t) = a0 2 + ∞ ∑ k=1 ak · cos (2 · π · k · f0 · t) + ∞ ∑ k=1 bk · sin (2 · π · k · f0 · t) = 2 2 + 5 · cos (2 · π · 1 · f0 · t) + 4 · sin (2 · π · 1 · f0 · t) − 2 · cos (2 · π · 2 · f0 · t) + 3 · sin (2 · π · 2 · f0 · t) + 1 · cos (2 · π · 3 · f0 · t) − 1 · sin (2 · π · 3 · f0 · t) = 1 + 5 · cos (2 · π · f0 · t) + 4 · sin (2 · π · f0 · t) − 2 · cos (4 · π · f0 · t) + 3 · sin (4 · π · f0 · t) + 1 · cos (6 · π · f0 · t) − 1 · sin (6 · π · f0 · t) x2(t) = A0 + ∞ ∑ k=1 Ak · cos (2 · π · k · f0 · t + αk) = 1 + 3 · cos (2 · π · 1 · f0 · t − π 3) + 2 · cos (2 · π · 3 · f0 · t + π 2) C o r r i g é d e s e x e r c i c e s, v 1 .1 6 2 3 M E E \ c o _ ts.te x \ 1 9 m a i 2 0 0 6 H E I G - V d T r a i t e m e n t d e S i g n a l (T S) x2(t) Le spectre unitlatéral correspond directement à l'expression de x2(t) en cosinus : x2(t) = A0 + ∞ ∑ k=1 Ak · cos (2 · π · k · f0 · t + αk) = 1 + 3 · cos (2 · π · 1 · f0 · t − π 3) + 2 · cos (2 · π · 3 · f0 · t + π 2) = A0 + A1 · cos (2 · π · 1 · f0 · t + α1) + A3 · cos (2 · π · 3 · f0 · t + α3) Le spectre bilatéral s'en déduit facilement : k 0 1 2 3 Ak A0 = 1 A1 = 3 A2 = 0 A3 = 2 αk α0 = 0 α1 = −π 3 α2 = 0 α3 = +π 2 k 0 ±1 ±2 ±3 X(j · k) X(j · 0) = A0 = 1 X(±j · 1) = A1 2 · e±j·α1 = 3 2 · e∓j·π 3 X(±j · 2) = 0 X(±j · 3) = A3 2 · e±j·α3 = 2 2 · e±j·π 2 = e±j·π 2 La représentation graphique des spectre uni- et bilatéraux est donnée sur la figure 1.10. x3(t) Le spectre unitlatéral correspond directement à l'expression de x3(t) en cosinus : x3(t) = 5 + 10 · cos (2 · π · 1 · f0 · t + 0.6435) + 2 · √ 5 · cos (2 · π · 3 · f0 · t + 2.6779) = A0 + A1 · cos (2 · π · 1 · f0 · t + α1) + A3 · cos (2 · π · 3 · f0 · t + α3) Le spectre bilatéral a dédjà été obtenu au précédemment : on avait : x3(t) = ∞ ∑ k=−∞ X(j · k) · ej·2·π·k·f0·t = X(−j · 3) · e−j·2·π·3·f0·t + X(−j · 1) · e−j·2·π·1·f0·t + X(j · 0) · ej·2·π·0·f0·t + X(j · 1) · e−j·2·π·1·f0·t + X(j · 3) · ej·2·π·3·f0·t = √ 5 · e−j·2.6779 · e−j·2·π·3·f0·t + 5 · e−j·0.6435 · e−j·2·π·1·f0·t + 5 + 5 · ej·0.6435 · ej·2·π·1·f0·t + √ 5 · ej·2.6779 · ej·2·π·3·f0·t C o r r i g é d e s e x e r c i c e s, v 1 .1 6 2 6 M E E \ c o _ ts.te x \ 1 9 m a i 2 0 0 6 H E I G - V d T r a i t e m e n t d e S i g n a l (T S) Si l'on répète néanmoins la même opération que pour x1(t) et x2(t), on a : k 0 1 2 3 Ak A0 = 5 A1 = 10 A2 = 0 A3 = 2 · √ 5 αk α0 = 0 α1 = 0.6435 α2 = 0 α3 = 2.6779 k 0 ±1 ±2 ±3 X(j · k) X(j · 0) = A0 = 5 X(±j · 1) = A1 2 · e±j·α1 = 10 2 · e±j·0.6435 = 5 · e±j·0.6435 X(±j · 2) = 0 X(±j · 3) = A3 2 · e±j·α3 = 2· √ 5 2 · e±j·2.6779 = √ 5 · e±j·2.6779 La représentation graphique des spectre uni- et bilatéraux est donnée sur la figure 1.11. C o r r i g é d e s e x e r c i c e s, v 1 .1 6 2 7 M E E \ c o _ ts.te x \ 1 9 m a i 2 0 0 6 HEIG-Vd Traitement de Signal (TS) • • • • 0 1000 2000 3000 0 2 4 6 fHz Ak • • • • 0 1000 2000 3000 −0.6 −0.4 −0.2 0 0.2 fHz αk π • • • • • • • −3000 −2000 −1000 0 1000 2000 3000 0 1 2 3 fHz |X(j · k)| • • • • • • • −3000 −2000 −1000 0 1000 2000 3000 −0.6 −0.4 −0.2 0 0.2 0.4 0.6 fHz arg{X(j·k)} π −2 −1 0 1 2 −5 0 5 10 t [ms] x(t) Fig. 1.9 – (fichier source). • • • • 0 1000 2000 3000 0 1 2 3 f [Hz] Ak • • • • 0 1000 2000 3000 −0.2 0 0.2 0.4 f [Hz] αk π • • • • • • • −3000 −2000 −1000 0 1000 2000 3000 0 0.5 1 1.5 f [Hz] |X(j · k)| • • • • • • • −3000 −2000 −1000 0 1000 2000 3000 −0.4 −0.2 0 0.2 0.4 f [Hz] arg{X(j·k)} π −2 −1 0 1 2 −4 −2 0 2 4 6 t [ms] x(t) Fig. 1.10 – (fichier source). Corrigé des exercices, v 1.16 28 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 1.1.8 Exercice SF 8 Considérant le signal x (t) = 2 + sin (2 · π · f0 · t) + 0.25 · cos (6 · π · f0 · t) 1. écrivez x (t) dans les formes cosinus et complexe ; 2. donnez les composantes spectrales dans les trois représentations : {ak, bk} {Ak, αk} {X (j · k)} 3. vérifiez que la puissance de ce signal calculée à l'aide des trois représenta- tions donne le même résultat ; 4. comment calculeriez-vous la puissance dans l'espace temps ? voyez-vous des moyens de simplifier ce calcul ? Si oui, le résultat est immédiat. Corrigé des exercices, v 1.16 31 MEE \co_ts.tex\19 mai 2006 H E I G - V d T r a i t e m e n t d e S i g n a l (T S) Corrigé 1. On a pour la série en cosinus : x (t) = 2 + sin (2 · π · f0 · t) + 0.25 · cos (6 · π · f0 · t) = 2 + cos (2 · π · f0 · t − π 2) + 0.25 · cos (6 · π · f0 · t) = A0 + A1 · cos (2 · π · f0 · t + α1) + A3 · cos (6 · π · f0 · t + α3) Partant de la série en cosinus, on obtient facilement la série complexe en faisant usage des formule d'Euler : x (t) = 2 + cos (2 · π · f0 · t − π 2) + 0.25 · cos (6 · π · f0 · t) = 2 + 1 2 · e+j·(2·π·f0·t− π 2) + 1 2 · e−j·(2·π·f0·t− π 2) + 0.25 1 2 · e+j·(6·π·f0·t) + 0.25 1 2 · e−j·(6·π·f0·t) = X (0) + X (+j · 1) · e+j·2·π·f0·t + X (−j · 1) · e−j·2·π·f0·t + X (+j · 3) · e+j·6·π·f0·t + X (−j · 3) · e−j·6·π·f0·t = 2 + 1 2 · e−j·π 2 · e+j·2·π·f0·t + 1 2 · e+j·π 2 · e−j·2·π·f0·t + 0.25 2 · e+j·6·π·f0·t + 0.25 2 · e−j·6·π·f0·t 2.
Série en cosinus : A0 = 2 A1 = 1 α1 = − π 2 A3 = 0.25 α3 = 0 C o r r i g é d e s e x e r c i c e s, v 1 .1 6 3 2 M E E \ c o _ ts.te x \ 1 9 m a i 2 0 0 6 H E I G - V d T r a i t e m e n t d e S i g n a l (T S) Série de Fourier : a0 = 2 · A0 = 2 · 2 = 4 a1 = +A1 · cos (α1) = cos (− π 2) = 0 b1 = −A1 · sin (α1) = − sin (− π 2) = 1 a3 = +A3 · cos (α3) = 0.25 · cos (0) = 0.25 b3 = −A3 · sin (α3) = −0.25 · sin (0) = 0 Série complexe : X(j · 0) = 2 X(±j · 1) = 0.5 · e∓j·π 2 X(±j · 3) = 0.125 · ej·0 3. Série de Fourier : P = (a0 2)2 + 1 2 · ∞ ∑ k=1 (√ a2 k + b2 k)2 = (a0 2)2 + 1 2 · [a2 1 + b2 1 + a2 3 + b2 3] = (4 2)2 + 1 2 · [02 + 12 + 0.252 + 02] = 4.52125 [V2] C o r r i g é d e s e x e r c i c e s, v 1 .1 6 3 3 M E E \ c o _ ts.te x \ 1 9 m a i 2 0 0 6 H E I G - V d T r a i t e m e n t d e S i g n a l (T S) Autrement dit, il suffit de sommer les carrés des valeurs efficaces de 2, sin (2 · π · f0 · t) et 0.25 ·cos (6 · π · f0 · t), soit : P = 22 + (1 √ 2)2 + 0.252 · (1 √ 2)2 = 4.53125 [V2] C o r r i g é d e s e x e r c i c e s, v 1 .1 6 3 6 M E E \ c o _ ts.te x \ 1 9 m a i 2 0 0 6 HEIG-Vd Traitement de Signal (TS) 1.1.9 Exercice SF 15 Considérant une SIR centrée de période T = 100 [µs], de largeur Δt = 20 [µs] et d'amplitude A = 10 [V], 1. calculez le pourcentage de puissance comprise dans le premier lobe du sinus cardinal ; 2. admettant que cette SIR est appliquée à un filtre passe-bas d'ordre 1 dont la fonction de transfert est H (j · f) = 1 1 + j · f fc fc = 10 [kHz] que valent l'amplitude et la phase des composantes 10 [kHz], 40 [kHz] et 150 [kHz] ? Corrigé La série de Fourier complexe d'une SIR a été calculée dans le cours. On a : X (j · k) = A · Δt T · sin (k · π · f0 · Δt) k · π · f0 · Δt = 10 [V] · 20 [µs] 100 [µs] · sin (k · π · 1 100 [µs] · 20 [µs]) k · π · 1 100 [µs] · 20 [µs] = 2 [V] · sin (k · π · 0.2) k · π · 0.2 Le spectre d'amplitude s'annule pour la première fois pour k = 5 (figure 1.12 page suivante). Le premier lobe du spectre est donc constitué des raies (pour mémoire, on a |X(j · k)| = |X(−j · k)|) X(−j · 5) = 0 X(−j · 4) = 0.4677 X(−j · 3) = 1.0091 X(−j · 2) = 1.5136 X(−j · 1) = 1.871 X(j · 0) = 2 X(j · 1) = 1.871 X(j · 2) = 1.5136 X(j · 3) = 1.0091 X(j · 4) = 0.4677 X(j · 5) = 0 Corrigé des exercices, v 1.16 37 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) −100 −50 0 50 100 0 0.5 1 1.5 2 f [kHz] |X(j · k)| −100 −50 0 50 100 −1 −0.5 0 0.5 1 f [kHz] arg{X(j·k)} π Fig. 1.12 – f0 = 1 100 [µs] = 10 [kHz] (fichier source). Corrigé des exercices, v 1.16 38 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 0 1 2 3 4 5 0 5 10 15 20 t [ms] u1(t) [V] 0 1 2 3 4 5 0 5 10 15 20 t [ms] u2(t) [V] 0 1 2 3 4 5 −0.02 −0.01 0 0.01 0.02 t [ms] i(t) [A] Corrigé des exercices, v 1.16 41 MEE \co_ts.tex\19 mai 2006 H E I G - V d T r a i t e m e n t d e S i g n a l (T S) Puissance du signal u1(t) La valeur DC n'est autre que la valeur moyenne X (j · 0) du signal. Partant de la définition de X(j · k) X (j · k) = 1 T · ∫ + T 2 − T 2 x (t) · e−j·2·π·k·f0·t · dt −∞ < k < +∞ on a, pour k = 0 : U1 (j · 0) = 1 T · ∫ + T 2 − T 2 x (t) · e−j·2·π·0·f0·t · dt = 1 T · ∫ + T 2 − T 2 x (t) · dt = 1 1 [ms] · ∫ + 1 [ms] 2 − 1 [ms] 2 x (t) · dt = 1 1 [ms] · ∫ + 1 [ms] 2 0 20 [V] · dt = 10 [V] La puissance AC se calcule par déduction de la puissance DC Pdc = |U1 (j · 0)|2 = 100 [V2] de la puissance totale C o r r i g é d e s e x e r c i c e s, v 1 .1 6 4 2 M E E \ c o _ ts.te x \ 1 9 m a i 2 0 0 6 H E I G - V d T r a i t e m e n t d e S i g n a l (T S) P . La puissance totale s'écrit : P = 1 T · ∫ + T 2 − T 2 x 2 (t) · dt = 1 1 [ms] · ∫ + 1 [ms] 2 − 1 [ms] 2 x 2 (t) · dt = 1 1 [ms] · ∫ + 1 [ms] 2 0 (20 [V])2 · dt = 1 1 [ms] · (20 [V])2 · [t] + 1 [ms] 2 0 = 1 1 [ms] · (20 [V])2 · [+ 1 [ms] 2 − 0] = 200 [V2] La valeur efficace est donc : U1eff = √ P = √ 200 [V2] = 10 · √ 2 [V] On déduit de la puissance totale P et de la puissance DC Pdc la puissance AC : P1,AC = P1 − P1,DC = 200 [V2] − 100 [V2] = 100 [V2] La valeur efficace AC est : U1effAC = √ P1,AC = 10 [V] Puissance du signal u2(t) La valeur DC sera égale à celle de u1(t) puisque l'on a affaire à un filtre passe-bas. Par calcul, C o r r i g é d e s e x e r c i c e s, v 1 .1 6 4 3 M E E \ c o _ ts.te x \ 1 9 m a i 2 0 0 6 H E I G - V d T r a i t e m e n t d e S i g n a l (T S) La puissance AC de u2(t) est dès lors : P2,AC = P2 − P1,DC = 200 [V2] − 100 [V2] = 100 [V2] La valeur efficace AC est : U2effAC = √ P2,AC = 10 [V] Puissance du signal i(t) Suite du corrigé en préparation. C o r r i g é d e s e x e r c i c e s, v 1 .1 6 4 6 M E E \ c o _ ts.te x \ 1 9 m a i 2 0 0 6 HEIG-Vd Traitement de Signal (TS) R Cx(t) y(t) Fig. 1.13 – (fichier source).
1.1.11 Exercice SF 17 Soit un filtre RC passe-bas dont la constante de temps est mal connue.
On lui applique une SIR x (t) d'amplitude A = 10 [V], de période T = 20 [ms] et de largeur Δt = 1 [ms]. 1. que valent les composantes continues des signaux d'entrée et de sortie ? 2. quelle est la fonction de transfert H (j · ω) du circuit ; 3. que valent les spectres bilatéraux X (j · k) et Y (j · k) ? 4. admettant que la constante de temps est de l'ordre de 2 [ms], esquissez les signaux d'entrée x (t) et de sortie y (t) ; estimez la valeur maximum de y (t) ; 5.
pour la fréquence f = 5 · f0, l'analyseur spectral du signal de sortie fournit le coefficient complexe Y (j · 5) = −0.0659 − j · 0.154 ; calculez l'amplitude et l'argument de la fonction de transfert pour cette fréquence ; (Rép. : |H| = 0.37,6 H = −68 [◦]) 6. que valent la constante de temps et la fréquence de coupure du filtre ? (Rép. : τ = 1.6 [ms], fc = 100 [Hz]) Corrigé 1. Comme il s'agit d'un filtre passe-bas (figure 1.13), la composante continue X(j · 0) de l'entrée x(t) se retrouve telle quelle à la sortie : Y (j · 0) = X(j · 0) 2. Sous l'hypothèse de régime sinusoïdal permanent, la fonction de transfert en j ·ω s'obtient en raisonnant avec des impédance complexes et en faisant usage de la règle du diviseur de tension : Y (j · k) = 1 j·ω·C R + 1 j·ω·C · X(j · k) Corrigé des exercices, v 1.16 47 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) d'où : H(j · ω) = Y (j · k) X(j · k) = 1 1 + j · ω · R · C = 1 1 + j · ω · τ = 1 1 + j · f fc | | | | | fc= 1 2·π·τ 3. Pour X(j · k), c'est le spectre bien connu d'une SIR : X(j·k) = A· Δt T ·sinc(k · f0 · Δt) = 10· 1 [ms] 20 [ms] ·sinc (k · 1 20 [ms] · 1 [ms]) = 0.5·sinc(0.05 · k) avec f0 = 1 T . Y (j · k) est donc simplement : Y (j ·k) = H(j · ω)|ω=2·π·f=2·π·k·f0 ·X(j ·k) = 1 1 + j · k · f0 fc ·0.5 · sinc(0.05 · k) Graphiquement, les résultats se présentent comme indiqué sur la figure 1.14. 4. La figure 1.15 montre le signal de sortie, obtenu ici non par calcul analytique (résolution de l'équation différentielle régissant le circuit de la figure 1.13) par synthèse à partie des N = 41 premiers termes de Y (j · k), i.e. par : yN (t) = +N ∑ k=−N Y (j · k) · e+j·2·π·k·f0·t 5. On a : H(j · ω)|ω=2·π·5·f0 = Y (j · 5) X(j · 5) = 0.0659 − j · 0.154 0.5 · sinc(0.05 · 5) = −0.1464 − 0.3421 · j = 0.3721 · e−j·113.1671 [◦] 6. Du point précédent, on tire : |H(j · ω)|ω=2·π·5·f0 = 0.3721 = | | | | | 1 1 + j · f fc | | | | | = 1 √ 1 + (f fc)2 Corrigé des exercices, v 1.16 48 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 1.1.12 Exercice SF 21 Un circuit non linéaire de type parabolique est modélisé par la caractéristique de transfert suivante : u2 (t) = α · u1 (t) + β · u2 1 (t) Sachant qu'on lui applique une tension sinusoïdale u1 (t) = A · sin (ω0 · t) : 1. déterminez les composantes spectrales que l'on obtient à la sortie ; 2. quelle est la puissance normalisée P2 du signal de sortie ? 3. que vaut-elle par rapport à celle du signal d'entrée P1 ? 4. faites l'A.N. avec A = 10 [V], ω = 2 · π · 100 [rad s] , α = 1, β = 0.2 [V−1] 5. esquissez u2 (t) ; quel est son taux de distorsion harmonique ? Corrigé 1. u2(t) a pour expression : u2 (t) = α · u1 (t) + β · u2 1 (t) = α · A · sin (ω0 · t) + β · (A · sin (ω0 · t)) 2 = α · A · sin (ω0 · t) + β · A2 · 1 − 2 · cos (2 · ω0 · t) 2 = β · A2 2 + α · A · sin (ω0 · t) − β · A2 2 · cos (2 · ω0 · t) Pour obtenir rapidement le spectre U2(j · k de u2(t), on peut dans ce cas faire usage des relations d'Euler : u2 (t) = β · A2 2 + α · A · ej·ω0·t − e−j·ω0·t 2 · j − β · A2 2 · ej·2·ω0·t + e−j·2·ω0·t 2 = β · A2 2 + α · A 2 · j · ej·ω0·t − α · A 2 · j · e−j·ω0·t − β · A2 4 · ej·2·ω0·t − β · A2 4 · e−j·2·ω0·t = 0.2 · 102 2 + 1 · 10 2 · j · ej·ω0·t − 1 · 10 2 · j · e−j·ω0·t − 0.2 · 102 4 · ej·2·ω0·t − 0.2 · 102 4 · e−j·2·ω0·t = 10 + 5 · e−j·π 2 · ej·ω0·t + 5 · e+j·π 2 · e−j·ω0·t + 5 · ej·π · ej·2·ω0·t + 5 · e−j·π · e−j·2·ω0·t = U2(j · 0) + U2(j · 1) + U2(−j · 1) + U2(j · 2) + U2(−j · 2) 2. La puissance du signal d'entrée est donnée par le carré de sa valeur efficace, soit P1 = U2 1eff = A √ 2 2 = A2 2 = 102 2 = 50 [V2] Corrigé des exercices, v 1.16 51 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Pour le signal de sortie u2(t), on a en faisant usage de Parseval : P2 = +∞ ∑ k=−∞ |X (j · k)|2 = |U2 (−j · 2)|2 + |U2 (−j · 1)|2 + |U2 (j · 0)|2 + |U2 (j · 1)|2 + |U2 (j · 2)|2 = | |5 · e−j·π | | 2 + | |5 · e−j·π 2 | | 2 + 102 + | |5 · e+j·π 2 | | 2 + | |5 · ej·π | | 2 = 52 + 52 + 102 + 52 + 52 = 200 [V2] 3. On a : P2 P1 = 200 [V2] 50 [V2] = 4 4. cf ci-dessus 5. Le taux de distortion harmonique (TDH) est donné par : TDH = Xeff (k > 1) Xeff (k = 1) = √ X2 (2) + X2 (3) + X2 (4) + . . . X2 (1) On a donc : TDH = √ U2 2 (2) U2 2 (1) = √ 52 52 = 100% Corrigé des exercices, v 1.16 52 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Chapitre 2 Analyse des signaux non périodiques 2.1 Corrigé des exercices 2.1.1 Exercice TF 1 À partir de la seule observation du signal temporel de la figure 2.1, précisez ce que vaut sa densité spectrale en f = 0 [Hz] puis calculez et esquissez sa transformée de Fourier. Corrigé Selon la propriété de la transformée de Fourrier X(0) = ∫ +∞ −∞ x(t) · dt on a : X(0) = 1 · 2 [ms] + 2 · 2 [ms] + 1 · 2 [ms] = 8 [ms] Le signal de la figure 2.1 page suivante est constitué de 3 impulsions rectangulaires décalées et pondérées.
Si y(t) est une impulsion rectangulaire définie comme y(t) = { 0 si |t| > Δt 2 1 si |t| ≤ Δt 2 dont la transformée de Fourier est Y (j · f) = F{y(t)} = Δt · sinc(f · Δt) alors x(t) peut être exprimé comme suit : x(t) = y (t + 4 [ms]) + 2 · y (t) + y (t − 4 [ms]) Corrigé des exercices, v 1.16 53 MEE \co_ts.tex\19 mai 2006 H E I G - V d T r a i t e m e n t d e S i g n a l (T S) + = Δt −0.4 −0.2 0 0.2 0.4 −1 −0.5 0 0.5 1 t [s] u (t) Δt −0.4 −0.2 0 0.2 0.4 −1 −0.5 0 0.5 1 t [s] u (t + Δt 2) Δt −0.4 −0.2 0 0.2 0.4 −1 −0.5 0 0.5 1 t [s] −u (t − Δt 2) −0.4 −0.2 0 0.2 0.4 −1 −0.5 0 0.5 1 t [s] x (t) = u (t + Δt 2) + u (t − Δt 2) −0.4 −0.2 0 0.2 0.4 −1 −0.5 0 0.5 1 t [s] 1 Δt · ∫ t −∞ u (τ + Δt 2) · dτ −0.4 −0.2 0 0.2 0.4 −1 −0.5 0 0.5 1 t [s] 1 Δt · ∫ t −∞ −u (τ − Δt 2) · dτ −0.4 −0.2 0 0.2 0.4 −1 −0.5 0 0.5 1 t [s] y(t) = 1 Δt · ∫ t −∞ x (τ) · dτ Fig. 2.3 – En traitillé l'intégrale sans prise en compte du facteur 1 Δt . C o r r i g é d e s e x e r c i c e s, v 1 .1 6 5 6 M E E \ c o _ ts.te x \ 1 9 m a i 2 0 0 6 HEIG-Vd Traitement de Signal (TS) y(t) correspond, à un facteur 1 Δt près, à l'intégrale de x(t) (figure 2.1.2) : y(t) = 1 Δt · ∫ t −∞ x(τ) · dτ Connaissant la propriété de la TF ∫ t −∞ x(τ) · dτ ←→ 1 j · 2 · π · f ·X(j · f) + 1 2 ·X(0) · δ(f) avec X(0) = ∫ +∞ −∞ x(t) · dt, l'on peut ainsi écrire : Y (j · f) = F{y(t)} = 1 Δt · 1 j · 2 · π · f ·X(j · f) + 1 2 · | +∞ −∞ x(t)·dt=0 X(0) ·δ(f) = 1 Δt · 1 j · 2 · π · f ·X(j · f) = 1 Δt · 1 j · 2 · π · f · A ·Δt · sinc(f ·Δt) · 2 · j · sin (π · f ·Δt) = A ·Δt · sinc(f ·Δt) · sin (π · f ·Δt) π · f ·Δt · = A ·Δt · sinc2(π · f ·Δt) Application numérique : Y (j · f) = A ·Δt · sinc2(π · f ·Δt) = 1 · 200 [ms] · sinc2(π · f · 200 [ms]) 2.1.3 Exercice TF 3 Partant de la TF d'une impulsion et d'un saut unité, trouvez celle de z(t) (figure 2.2). Est-il possible de trouver Z(j · f) à partir de Y (j · f) ? Vous pouvez vérifier votre résultat en calculant Z(j · f = 0) qui doit être égal à Δt 2 . Corrigé 1. z(t) correspond à la somme (figure 2.4) de (a) l'intégrale d'une impulsion rectangulaire v(t) de largeur Δt, retardée de Δt 2 et d'amplitude − 1 Δt : (b) et d'un saut unité ϱ(t) Corrigé des exercices, v 1.16 57 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Δt −0.4 −0.2 0 0.2 0.4 −5 −4 −3 −2 −1 0 1 t [s] v (t) −0.4 −0.2 0 0.2 0.4 −1 −0.5 0 0.5 1 t [s] ∫ t −∞ v (τ) · dτ −0.4 −0.2 0 0.2 0.4 −1 −0.5 0 0.5 1 t [s] ϱ (t) −0.4 −0.2 0 0.2 0.4 −1 −0.5 0 0.5 1 t [s] z (t) = ∫ t −∞ v (τ) · dτ + ϱ(t) Fig. 2.4 – Corrigé des exercices, v 1.16 58 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) −4 −2 0 2 4 −1 −0.5 0 0.5 1 t [s] x(t) Fig. 2.5 – Le signal carré périodique x(t) s'exprime partant de sa série de Fourier com- plexe X(j · k) : x (t) = ∞∑ k=−∞ X (j · k) · e+j·2·π·k·f0·t La transformée de Fourier de x(t) s'écrit alors, en appliquant la définition et en tenant compte de la transformée de Fourier d'un phaseur : X(j · f) = ∫ +∞ −∞ x(t) · e−j·2·π·f·t · dt = ∫ +∞ −∞ ∞∑ k=−∞ X (j · k) · e+j·2·π·k·f0·t · e−j·2·π·f ·t · dt = ∞∑ k=−∞ X (j · k) · ∫ +∞ −∞ e+j·2·π·k·f0·t · e−j·2·π·f·t phaseur · e−j·2·π·f·t · dt = ∞∑ k=−∞ X (j · k) · δ(f − k · f0) On obtient donc bel et bien un spectre de raies, représentées par des impulsions de Dirac pondérées par X(j · k) (figure 2.7). Corrigé des exercices, v 1.16 61 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) −10 −5 0 5 10 0 0.2 0.4 0.6 f [Hz] |X(j · k)| −10 −5 0 5 10 −1 −0.5 0 0.5 1 f [Hz] arg{X(j·k)} π Fig. 2.6 – −10 −5 0 5 10 0 0.2 0.4 0.6 f [Hz] |X(j · f)| −10 −5 0 5 10 −1 −0.5 0 0.5 1 f [Hz] arg{X(j·f)} π Fig. 2.7 – Corrigé des exercices, v 1.16 62 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 2.1.5 Exercice TF 5 Considérant le signal x(t) = e−a·|t|, calculez et esquissez x(t) et X(j · f), puis vérifiez les 2 égalités suivantes : X(0) = ∫ +∞ −∞ x(t) · dt x(0) = ∫ +∞ −∞ X(j · f) · df Corrigé En préparation. 2.1.6 Exercice TF 6 fréquence temps 1 la partie réelle de X(j · f) est nulle 2 la partie imaginaire de X(j · f) est nulle 3 il existe un décalage t0 tel que ej·2·π·f ·t0 · X(j · f) est réel 4 X(j · f) est continu 1. Considérant les quatre propriétés fréquentielles du tableau ci-dessus, expri- mez leur équivalent temporel dans la colonne de droite. 2. Pour chacun des signaux temporels de la figure 2.8, quelles sont les proprié- tés du tableau qui s'y appliquent ? 3.
Construisez un signal qui ne possède aucune des quatre propriétés mention- nées dans le tableau. Corrigé En préparation. Corrigé des exercices, v 1.16 63 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Corrigé En préparation.
2.1.9 Exercice TF 9 On applique une exponentielle décroissante u1(t) = U0·e −a·t ·ϱ(t), d'amortissement a = 100 [s−1] à un filtre passe-bas de constante de temps τ = 1 [ms] ; 1. calculez la TF U2(j · f) de la tension de sortie u2(t) du filtre ; 2. utilisez le tableau des transformées pour déduire l'expression temporelle de u2(t). Corrigé 1. La fonction de transfert du filtre, exprimée dans le domaine fréquentiel, est : H(j · f) = U2(j · f) U1(j · f) = 1 1 + j · 2 · π · f · τ On a donc, en tenant compte du fait que la transformée de Fourier de u1(t) a été calculée au §2.3.1 : U2(j · f) = H(j · f) · U1(j · f) = 1 1 + j · 2 · π · f · τ · U1(j · f) = 1 1 + j · 2 · π · f · τ · U0 · 1 a + j · 2 · π · f = U0 τ · 1 1 τ + j · 2 · π · f · 1 a + j · 2 · π · f 2. La transformée de Fourier inverse fournit directement u2(t) (annexe §2.A) : u2(t) = U0 τ · 1 a − 1 τ · (e− t τ − e−a·t) · ϱ(t) = U0 a · τ − 1 · (e− t τ − e−a·t) · ϱ(t) = U0 1 − a · τ · (e−a·t − e− t τ) · ϱ(t) 2.1.10 Exercice TF 10 Soit un message m(t) = A · cos(2 ·π ·f1 · t) modulé en amplitude par une porteuse sinusoïdale p(t) = sin(2 · π · f0 · t) : Corrigé des exercices, v 1.16 66 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 1. calculez la TF du signal modulé x(t) = m(t) · p(t) = A · sin(2 · π · f0 · t) · cos(2 · π · f1 · t) ; 2. esquissez le spectre du signal modulé |X(j · f)| si f1 = 10 [kHz] et f0 = 800 [kHz] ; 3. idem que le point 2) lorsque le signal m(t) possède un spectre continu |M(j · f)| triangulaire et non-nul entre 2 [kHz] et 10 [kHz]. Corrigé En préparation. 2.1.11 Exercice TF 11 Soit le signal : u(t) = { U0 · cos(2 · π · f0 · t) si |t| ≤ t0 0 si |t| > t0 1. esquissez u(t) ; 2. calculez sa TF U(j · f) ; 3. esquissez |U(j · f)| pour U0 = 1 [V] T = 1 f0 = 1 [ms] t0 = 10 [ms]. Ce signal correspond à l'observation d'une fonction sinusoïdale pendant une durée finie 2 ·t0.
On remarquera, une fois le calcul effectué, que l'analyse spectrale d'une sinusoïde pendant une durée finie revient à remplacer les raies spectrales situées en f = ±f0 par la fonction sinus cardinal. Corrigé On sait que 1. 2. On peut exprimer u(t) comme u(t) = rect (t, 2 · Δt)·U0·cos(2·π·f0·t) = rect (t, 2 · Δt)·U0· ej·2·π·f0·t + e−j·2·π·f0·t 2 où la fonction rect (t, Δt) est définie comme suit : rect (t, Δt) = { 0 si |t| > Δt 2 1 si |t| ≤ Δt 2 Sa TF est (§2.2.1) Δt · sinc (π · f · δt) Corrigé des exercices, v 1.16 67 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Sachant que (§2.1.4, propriété de modulation) x(t) · e+j·2·π·f0·t ←→ X (j · (f − f0)) on peut écrire : U(j ·ω) = U0 ·2 ·Δt ·sinc (π ·Δt · (f − f0))+U0 ·2 ·Δt ·sinc (π ·Δt · (f + f0)) 3. 2.1.12 Exercice TF 12 Soit la fonction : u(t) = { 1 2 · [1− cos(2 · π · f0 · t)] si |t| ≤ T 2 0 si |t| > T 2 1. esquissez u(t) ; 2. calculez sa TF U(j · f) ; 3. esquissez U(j · f) et la TF d'une impulsion rectangulaire de même durée ; 4. observez les différences. Corrigé En préparation. 2.1.13 Exercice TF 13 Connaissant la transformée E(j · f) d'un saut unité ϱ(t), calculez la transformée S(j · f) de la fonction signe s(t). Corrigé En préparation. 2.1.14 Exercice TF 14 Montrez qu'un produit simple dans l'espace des fréquences correspond à un pro- duit de convolution dans l'espace temps : Y (j · f) = X(j · f) ·H(j · f) ⇔ y(t) = x(t) ∗ h(t) = ∫ +∞ −∞ x(θ) · h(t− θ) · dθ Corrigé des exercices, v 1.16 68 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 1. esquisser U(j · f) ; 2. calculer puis esquissez u(t) ; 3. que vaut sa puissance ?
Corrigé 1. 2. En profitant de la propriété de symétrie F {x(t)} = X(j · f) ⟺ F−1 {x(j · f)} = X(−t) et en expimant X(j · f) sous la forme de 2 impulsions décalées dans le domaine des fréquences, X(j · f) = rect f + f0 150 [Hz] , Δf 100 [Hz] + rect (f − f0, Δf) on a : x(t) = Δf · sinc(π · Δf · (−t)) · e−j·2·π·f0·t + Δf · sinc(π · Δf · (−t)) · e+j·2·π·f0·t = 2 · Δf · sinc(π · Δf · t) · cos (2 · π · f0 · t) 3. La puissance de x(t) est avantageusement calculée dans l'espace des fré- quences : Wx = ∫ +∞ −∞ Sx(f) · df = ∫ +∞ −∞ |X(j · f)|2 · df = 1 · Δf + 1 · Δf = 2 · Δf 2.1.17 Exercice TF 17 Utiliser la transformation de Fourier pour trouver le courant circulant dans un circuit RC série sachant que le signal appliqué est un saut de tension d'amplitude E. Corrigé des exercices, v 1.16 71 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Corrigé Le circuit est décrit par l'équation différentielle (conditions initiales nulles) : u(t) = R · i(t) + 1 C · ∫ t −∞ i(τ) · dτ La transformée de Fourier des 2 membres de cette équation différentielle donne : U(j · f) = R · I(j · f) + 1 C · (1 j · 2 · π · f · I(j · f) + 1 2 · δ(f) · I(0)) = R · I(j · f) + 1 C · I(j · f) + 0∀f 1 2 · δ(f) · I(0)) · j · 2 · π · f = R · I(j · f) + 1 C · I(j · f) j · 2 · π · f = (R + 1 C · 1 j · 2 · π · f) · I(j · f) = j · 2 · π · f · R · C + 1 j · 2 · π · f · C · I(j · f) On en déduit : I(j · f) = j · 2 · π · f · C 1 + j · 2 · π · f · R · C · U(j · f) = j · 2 · π · f 1 + j · 2 · π · f · R · C · (1 j · 2 · π · f · E + 1 2 · δ(f) · E) = C 1 + j · 2 · π · f · R · C · j · 2 · π · f j · 2 · π · f · E + 1 2 · δ(f) · E 0 = C 1 + j · 2 · π · f · R · C · E De façon à être compatible avec les formes de présentation des transformées de Fourier utilisées dans les tables, on s'arrange pour que les coefficients des plus haute puissance de j · ω (ici 1 au dénominateur, 0 au numérateur) soient uni- Corrigé des exercices, v 1.16 72 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) taires : I(j · f) = C R·C j · 2 · π · f + 1 R·C · E = 1 R j · 2 · π · f + 1 R·C · E = 1 R · 1 j · 2 · π · f + 1 R·C · E En se référant à l'annexe 2A, on a, avec a = 1 R·C : i(t) = F−1 {I(j · f)} = E R · e− 1 R·C ·t · ϱ(t) 2.1.18 Exercice TF 18 On applique une fonction signe u1(t) d'amplitude E à un filtre RC passe-bas. 1. utilisez la transformation de Fourier pour trouver la tension de sortie ; 2. esquissez u1(t) et u2(t). Corrigé En préparation. 2.1.19 Exercice TF 19 On applique une exponentielle symétrique u1(t) = U0 · e −a·|t| à un filtre passe-bas de constante de temps τ . 1. avant de vous lancer dans les calculs, esquissez u1(t) et imaginez ce que peut être u2(t) ; 2. calculez la tension de sortie du filtre. La marche à suivre est la même que celle utilisée avec la transformation de La- place : décomposition en somme de fractions simples puis recherche des coeffi- cients par identification avec des transformées connues. Corrigé En préparation. Corrigé des exercices, v 1.16 73 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) −4 −3 −2 −1 0 1 2 3 4 −5 0 5 10 15 20 x 10 −4 fréquence [kHz] X (j f) 1/21/2 Fig. 2.12 – Exercice TF24. Corrigé En préparation. 2.1.25 Exercice TF 25 A partir du signal x(t) = e−a·t · ϱ(t), trouvez le spectre de y(t) = sgn(t). Corrigé En préparation. 2.1.26 Exercice Corr 1 Considérant le signal x(t) défini comme suit : x(t) = −A si −Δt < t < 0 0 si t = 0 +A si 0 < t < Δt 0 si |t| ≥ Δt on demande : 1. esquissez x(t) 2. calculez sa fonction d'autocorrélation pour les valeurs particulières suivantes τ = 0 τ = ±Δt τ = ±2 · Δt 3. esquissez la fonction rxx(τ) −∞ < τ < +∞. Corrigé des exercices, v 1.16 76 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Corrigé 1. L'esquisse de x(t) est présenté à la figure 2.13(a). 2. – Pour τ = 0 [s], la fonction d'autocorrélation est rxx(0) = Wx = ∫ +∞ −∞ x(t) · x(t) · dt = ∫ +∞ −∞ x 2(t) · dt On a : rxx(0) = 2 · A2 · Δt – Pour τ = Δt, la situation est décrite sur la figure 2.13(b), avec en gris la surface définie par le produit x(t) · x(t + τ). Comme, par définition, la fonction d'autocorrélation est l'intégrale de ce produit, i.e. rxx(τ) = ∫ +∞ −∞ x(t) · x(t + τ) · dt on a : rxx(Δt) = −A2 · Δt – Pour τ = 2 ·Δt, la situation est décrite sur la figure 2.13(c). On a claire- ment : rxx(2 · Δt) = 0 3. Comme les surfaces définies par le produit x(t)·x(t+τ) évoluent linéairement avec τ et que l'on dispose des valeurs de rx(τ) pour τ = 0 [s], τ = Δt et τ = 2 · Δt, on peut facilement esquisser rx(τ) (figure 2.14). Corrigé des exercices, v 1.16 77 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) ΔtΔt +A −A −1 −0.5 0 0.5 1 −4 −2 0 2 4 t [s] x(t) (a) x(t) ΔtΔt τ A −1 −0.5 0 0.5 1 −4 −2 0 2 4 t [s] x(t), x(t + τ) (b) τ = Δt ΔtΔt τ A −1 −0.5 0 0.5 1 −4 −2 0 2 4 t [s] x(t), x(t + τ) (c) τ = 2 · Δt ΔtΔt τ A −1 −0.5 0 0.5 1 −4 −2 0 2 4 t [s] x(t), x(t + τ) (d) τ = 0.33 [s] Fig. 2.13 – (a) : signal x(t).(b) : décalage de τ = Δt. (c) : décalage de τ = 2 · Δt. (d) : décalage de τ = 0.33 [s] (fichier source). Corrigé des exercices, v 1.16 78 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Chapitre 3 Echantillonnage des signaux analogiques 3.1 Corrigé des exercices 3.1.1 Exercice ECH 1 Considérant un signal dont le spectre est représenté à la figure 3.1, déterminez la fréquence d'échantillonnage minimum pour qu'il n'y ait pas de recouvrement spectral. Admettant fe = 16 [kHz], 1. dessinez le spectre du signal échantillonné pour f compris entre ±16 [kHz] ; 2.
que faut-il faire pour éviter le recouvrement spectral ? 3. dessinez le nouveau spectre ; quel en est l'avantage ? Corrigé On constate que le spectre proposé est borné par fmax = 10 [kHz] ; la fréquence d'échantillonnage devrait donc être supérieure à 2 · fmax = 20 [kHz]. 1.
Cependant, comme on propose fe = 16 [kHz], il y aura inévitablement du recouvrement spectral pour f > fe − fmax = 6 [kHz].
2. En filtrant analogiquement le signal temporel avant de l'échantillonner, on pourra supprimer les fréquences supérieures à fN = fe 2 = 8 [kHz] et éviter ainsi tout recouvrement jusqu'à la fréquence de Nyquist fN . 3. On a ainsi gagné 2 [kHz] de bande passante non perturbée par le recouvre- ment spectral. Corrigé des exercices, v 1.16 81 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) −20 −15 −10 −5 0 5 10 15 20 0 0.02 0.04 0.06 0.08 0.1 0.12 f [kHz] X (j f) [V /H z] f_xechant1_1.eps Fig. 3.1 – Exercice 1 (fichier source). 3.1.2 Exercice ECH 2 On échantillonne un signal xa(t) = cos(2 · π · 1000 · t) ; 1. esquissez xa(t) sur 3 périodes T au moins puis échantillonnez xa(t) avec (a) Te = T 4 ; (b) Te = T 2 ; (c) Te = 3 · T 4 ; 2. esquissez Xa(j · f) ; 3.
esquissez les 3 spectres Xe(j · f) correspondant aux 3 échantillonnages ; analysez et commentez. Corrigé En préparation. 3.1.3 Exercice ECH 3 On considère une SIR d'amplitude A = 10 [V], de période T0 = 1 [ms] et de largeur Δt = T0 4 que l'on échantillonne avec Te = T0 20 ; 1. esquissez x(t) et xe(t) ; 2. esquissez X(j · f) et Xe(j · f) ; 3.
que valent X(j · f) et Xe(j · f) pour f = 3 [kHz] ? Rép. : Xe(+j · 3) = X(+j · 3) + X(−j · 17) + X(+j · 23) Corrigé des exercices, v 1.16 82 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Corrigé 1. En préparation. 2. La densité spectrale d'amplitude d'une SIR a été calculée au chap.1 : X (j · k) = A · Δt T · sin (k · π · f0 · Δt) k · π · f0 · Δt Comme les raies spectrales des séries de Fourier complexes X(j · k) de- viennent des impulsions de Dirac lorsque l'on passe à la transformée de Fourier (cf.ex.TF4), on a pour X(j · f) : X (j · f) = +∞∑ k=−∞ A · Δt T · sin (k · π · f0 · Δt) k · π · f0 · Δt · δ(f − k · f0) Par suite de l'échantillonnage, cette densité spectrale va se répéter et accu- muler tous les fe, i.e. dans le cas particulier tous les fe = 20 · f0 = 20 [kHz]. Xe (j · f) = +∞∑ m=−∞ X(j · (f − m · fe)) (3.1) Comme X(j · f) n'est pas à bande limitée, il y aura recouvrement spectral, i.e. on retrouvera dans la bande de base −fe 2 . . . + fe 2 = −10 [kHz] . . . + 10 [kHz] des raies spectrales correspondant à des fréquences du spectre ori- ginal X(j · f) supérieures à fe 2 = 10 [kHz]. 3. Pour la raie spectrale de Xe(j · f) située à f = 3 [kHz], on aura : – La raie originale de X(j · f), correspondant à m = 0 et k = 3 dans l'expression (3.1) ci-dessus ; – La raie de Xe(j · f), correspondant à m = 1 et k = −17 de (3.1) ; – La raie de Xe(j · f), correspondant à m = −1 et k = +23 de (3.1) ; On aura donc, en négligeant le recouvrement pour |m| > 1 : Xe(j · 3 [kHz]) = X(j · 3 [kHz]) + X(−j · 17 [kHz]) + X(+j · 23 [kHz]) 3.1.4 Exercice ECH 4 Soit un signal en dents de scie d'amplitude A = 5 [V], de période T0 = 1 [ms] que l'on échantillonne avec la fréquence fe = 8 [kHz] ; 1. esquissez x(t) et xe(t) ; 2. sachant que X(j · k) = (−1)k+1 · A j·k·π , esquissez X(j · f) et Xe(j · f) ; 3. que valent X(j · f) et Xe(j · f) pour f = 1 [kHz] ?
Corrigé des exercices, v 1.16 83 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Corrigé En préparation.
3.1.10 Exercice ECH 10 On échantillonne un signal sinusoïdal d'amplitude 5 [V] avec un CAN 16 [bit] ±10 [V] en- tâché d'une de non-linéarité de ±1 2 LSB. Est-il possible de garantir un SNR d'au moins 90 [dB] ? Corrigé En préparation. 3.1.11 Exercice ECH 11 On échantillonne un signal analogique x(t) = 4 · cos(2 · π · 300 · t) + 2 · cos(2 · π · 900 · t) [V] avec un convertisseur AN 16 bits travaillant entre + et −5 [V] qui possède une non linéarité de ±1 2 LSB.
Les valeurs numériques du CAN sont transmises à travers une ligne dont le débit est de 104 [oct s] . On demande : 1. y a-t-il repliement spectral ? 2. que valent la résolution et le pas de quantification du convertisseur ? 3.
que vaut la puissance du signal x(t) ? quelle est sa valeur efficace ? 4. que vaut le rapport signal sur bruit de conversion AN ? Corrigé En préparation. 3.1.12 Exercice ECH 12 On utilise un filtre analogique passe-bas de Butterworth d'ordre 6 et de fréquence de coupure 4 [kHz] comme filtre antirepliement. Considérant que le signal échan- tillonné est perturbé par une composante spectrale d'amplitude A = 5 [V] et de fréquence f0 = 8 [kHz], on demande : 1.
quelle fréquence d'échantillonnage choisissez-vous pour que le repliement de la perturbation se fasse en f ≥ fc ?
2. quelle sera l'amplitude Ar du signal replié en f = fc ? Corrigé des exercices, v 1.16 86 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) Corrigé En préparation. 3.1.13 Exercice ECH 13 On utilise un filtre analogique passe-bas de Butterworth d'ordre 3 (sa fréquence de coupure fc est fixée par l'application) comme filtre antirepliement en amont d'un convertisseur AN 12 bits avec ± 1 2 LSB de non linéarité. 1. quelle est la résolution du convertisseur comprenant la quantification et la non-linéarité ; 2. esquissez la réponse fréquentielle du filtre et celle causée par le repliement spectral ; 3.
calculez la fréquence d'échantillonnage nécessaire pour que l'affaiblissement du repliement spectral en f = fc soit inférieur à la résolution du convertis- seur. Rép. : fe = 13.7 · fc Corrigé En préparation. 3.1.14 Exercice ECH 14 Un signal x(t) sinusoïdal d'amplitude A = 10 [V] de fréquence f = 1 [kHz] est échantillonné très rapidement (à 1 [MHz], par exemple) à l'aide d'un convertisseur analogique-numérique 4 bits travaillant entre + et −10 [V]. 1. esquissez les signaux x(t), xe[n], xq(t) ; 2. esquissez l'erreur de quantification e(t) ; 3. quelle est la valeur efficace de ce bruit de quantification ? 4. que vaut le SNR? Corrigé 1. Voir figure 3.2 page suivante 2.
Voir figure 3.2 page suivante 3. Selon le cours, la puissance du bruit de quantication est donnée par : PQ = Q2 12 Corrigé des exercices, v 1.16 87 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1 x 10 −3 −10 −5 0 5 10 temps A m p lit u d e Quantification d'un signal analogique original codage Bruit 0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1 x 10 −3 −1 −0.5 0 0.5 1 temps A m p lit u d e Quantification d'un signal analogique. Q eff =0.34541[V] Bruit f_ex_ECH_14_3.eps Fig. 3.2 – (fichier source) Corrigé des exercices, v 1.16 88 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) on voit que l'on a 1 a + j · 2 · π · (f − m · fe) | | | | m<0 + 1 a + j · 2 · π · (f − m · fe) | | | | m>0 = 1 a + j · 2 · π · (f + m · fe) | | | | m>0 + 1 a + j · 2 · π · (f − m · fe) | | | | m>0 = 1 (a + j · 2 · π · f) + j · 2 · π · m · fe + 1 (a + j · 2 · π · f) − j · 2 · π · m · fe = (a + j · 2 · π · f) − j · 2 · π · m · fe + (a + j · 2 · π · f) + j · 2 · π · m · fe (a + j · 2 · π · f)2 − (j · 2 · π · m · fe)2 = 2 · (a + j · 2 · π · f) (a + j · 2 · π · f)2 − (j · 2 · π · m · fe)2 = 2 · (a + j · 2 · π · f) (a + j · 2 · π · f)2 + (2 · π · m · fe)2 Ce qui donne finalement Xe(j · f) = 1 a + j · 2 · π · f + 2 · +∞ ∑ m=1 a + j · 2 · π · f (a + j · 2 · π · f)2 + (2 · π · m · fe) 2 3.1.18 Exercice ECH 18 Considérant un signal carré à valeur moyenne nulle de période T0 = 1 [ms] et d'amplitude A = 1 [V] que l'on échantillonne à la fréquence fe = 9.8 [kHz], on demande : 1. Quelles sont les fréquences et amplitudes des raies spectrales du signal ana- logique ? Esquissez le spectre d'amplitudes. 2. Quelle est la largeur de la bande de base ? Quelles sont les composantes spectrales réelles présentes dans la bande de base ? 3. Quelles sont les fréquences apparentes d'ordre n ∈ [0, . . . , 15] présentes dans la bande de base ? 4. Quelles sont les amplitudes de chacune de ces raies ? 5. Les résultats de l'analyse spectrale sont donnés dans la figure 3.4 ; asso- ciez les numéros des composantes spectrales théoriques aux raies spectrales obtenues après échantillonnage. Corrigé 1. Le spectre d'un signal carré d'amplitude A est décrit par X(j·k) = A· Δt T · sin (k · π · f0 · Δt) k · π · f0 · Δt = A 2 · sin (k · π 2) k · π 2 = { 0 si k pair ou k = 0 ± A k·π si k impair Corrigé des exercices, v 1.16 91 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 0 1 2 3 4 5 6 7 8 9 10 −1 −0.5 0 0.5 1 Signal échantillonné x e (t) temps [ms] x (t) 0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 −40 −30 −20 −10 0 Spectre théorique (o) et spectre dû au repliement spectral (−) fréquence [kHz] |X (j f) | [d B] f 0 = 1 f_xechcarre_1.eps Fig. 3.3 – Echantillonnage et repliement spectral pour un signal carré (fichier source). Corrigé des exercices, v 1.16 92 MEE \co_ts.tex\19 mai 2006 HEIG-Vd Traitement de Signal (TS) 0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 −40 −35 −30 −25 −20 −15 −10 −5 0 Spectre théorique (o) et spectre dû au repliement spectral (−) fréquence [kHz] |X (j f) | [d B] f 0 = 1 f N 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 353739 41 43 45 f_cx_echcarre_1.eps Fig.
3.4 – Repliement spectral pour un signal carré. 2. La bande de base est comprise entre ±fe 2 = 4.9 [kHz]. Les composantes spectrales réelles présentes se situent donc en ±1 kHz et ±3 [kHz]. 3. Les fréquences apparentes sont présentées dans la figure 3.4. 4. L'amplitude de chaque raie d'ordre k vaut A k·π . 5. Voir figure 3.4 Corrigé des exercices, v 1.16 93 MEE \co_ts.tex\19 mai 2006 Entre un fichier exécutable sur le disque dur et un programme en cours d'exécution, il y a un monde. Le fichier exécutable doit être copié en mémoire RAM par le chargeur de programme, le système d'exploitation doit gérer tout ce qui a trait à l'allocation mémoire, configurer certaines de ses structures et j'en passe et des meilleures. Le résultat final de ces opérations est ce qu'on appelle un processus, un programme chargé en mémoire et en cours d'exécution sur l'ordinateur. Pour manipuler des processus, l'OS doit mémoriser des informations sur eux : à quel fichier exécutable correspond le processus, où se situe-t-il en mémoire et ainsi de suite. Toutes ces informations sont stockées dans ce qu'on appelle un Process Control Block, une portion de la mémoire dans laquelle le système d'exploitation va stocker toutes les informations attribuées à un processus.
Vie et mort des processus Ordonnancement Communication entre processus Vie et mort des processus Création de processus Les processus peuvent naître et mourir. La plupart des processus démarre quand l'utilisateur demande l'exécution d'un programme, n'importe quel double-clic sur une icône d'exécutable en est un bon exemple. Mais, le démarrage de la machine en est un autre exemple.
En effet, il faut bien lancer le système d'exploitation, ce qui demande de démarrer un certain nombre de processus. Après tout, vous avez toujours un certain nombre de services qui se lancent avec le système d'exploitation et qui tournent en arrière-plan. À l'exception du premier processus, lancé par le noyau, les processus sont toujours créés par un autre processus. On dit que le processus qui les a créés est le processus parent. Les processus créés par le parent sont appelés processus enfants. On parle aussi de père et de fils. Certains systèmes d'exploitation conservent des informations sur qui a démarré quoi et mémorisent ainsi qui est le père ou le fils pour chaque processus. L'ensemble forme alors une hiérarchie de processus. Sur la majorité des systèmes d'exploitation, un appel système suffit pour créer un processus. Mais certains systèmes d'exploitation (les unixoïdes, notamment) ne fonctionnent pas comme cela. Sur ces systèmes, la création d'un nouveau processus est indirecte : on doit d'abord copier un processus existant avant de remplacer son code par celui d'un autre programme. Ces deux étapes correspondent à deux appels systèmes différents. Destruction de processus Si les processus peuvent être créés, il est aussi possible de les détruire. De façon générale, un processus peut se terminer de deux façons : de façon normale (il a terminé son exécution) ; de façon anormale (le processus a commis une « faute », par exemple en tentant d'accéder à une zone mémoire protégée ou en réalisant une division par zéro, ou s'est trouvé en présence d'un problème ou d'une erreur l'empêchant de poursuivre). Dans tous les cas, le processus père est informé de la terminaison du processus fils par le système d'exploitation. Le processus fils passe alors dans un mode appelé mode zombie : il attend que son père prenne connaissance de sa terminaison. Lorsque le processus se termine, le système récupère tout ce qu'il a attribué au processus (PCB, espaces mémoire, etc.), mais garde une trace du processus dans sa table des processus. C'est seulement lorsque le père prend connaissance de la mort de son fils qu'il supprime l'entrée du fils dans la table des processus. Et si un processus père est terminé avant son fils, il se passe quoi ? Bonne question ! Nous avons dit plus haut qu'un processus devait toujours avoir un père.
C'est le processus qui est le père de tous qui « adopte » le processus qui a perdu son père et qui se chargera alors de le « tuer ». Ordonnancement Pouvoir lancer plusieurs programmes en même temps est quelques chose de commun. Mais tout cela doit pouvoir fonctionner avec des ordinateurs qui ne sont pourvu que d'un seul processeur. Or, un processeur ne peut exécuter qu'un seul programme à la fois (il existe toutefois des exceptions). Pour éviter tout problème, le système d'exploitation utilise une technique pour permettre la multiprogrammation sur les ordinateurs à un seul processeur : l'ordonnancement. Cela consiste à constamment switcher entre les différents programmes qui doivent être exécutés. Ainsi, en switchant assez vite, on peut donner l'illusion que plusieurs processus s'exécutent en même temps. Image de Raphael Javaux, GFDL et CC-BY-SA 3.0, wikicommons États d'un processus Tous les programmes n'ont pas forcément besoin du processeur, certains attendent par exemple la réponse d'un périphérique et ne peuvent donc rien faire durant cette période.
Pour éviter de les exécuter inutilement, l'OS se souvient que ces processus sont bloqués en attente des entrées-sorties. Même chose pour les programmes qui attendent le processeur : l'OS doit savoir quel est le programme en cours d'exécution et quels sont ceux qui attendent. Pour cela, on va donner à chaque processus un état qui permettra de savoir si notre programme veut s'exécuter sur le processeur. Un processus peut être dans (au moins) trois états : Élu : un processus en état élu est en train de s'exécuter sur le processeur. Prêt : celui-ci a besoin de s'exécuter sur le processeur et attend son tour. Bloqué : celui-ci n'a pas besoin de s'exécuter (par exemple, parce que celui-ci attend une donnée en provenance d'une entrée-sortie). Il arrive qu'un programme stoppe son exécution. Il peut y avoir plusieurs raisons à cela : il a besoin d'accéder à un périphérique et passe donc à l'état bloqué ; il n'a plus rien à faire et a fini (temporairement ou définitivement) son exécution ; il est interrompu par le système d'exploitation, histoire de laisser la place à un autre programme. Ordonnancement Sélection Lors d'un changement d'état, le processus en cours d'exécution passe en état bloqué ou prêt. En même temps, un processus va passer de l'état prêt à l'état élu. Reste qu'il faut choisir un programme dans la liste d'attente et c'est le rôle d'un programme système spécialisé : l'ordonnanceur. Pour utiliser notre processeur au mieux, il faut faire en sorte que tous les processus aient leur part du gâteau et éviter qu'un petit nombre de processus monopolisent le processeur.
Bien choisir le processus à exécuter est donc une nécessité. On peut préciser qu'il existe deux grandes formes d'ordonnancements : l'ordonnancement collaboratif et l'ordonnancement préemptif. Dans le premier cas, c'est le processus lui-même qui décide de passer de l'état élu à l'état bloqué ou prêt. Pour ce faire, le programme en question doit exécuter un appel système bien précis qui rendra la main à l'ordonnanceur. Dans ces conditions, il n'est pas rare qu'un processus « un peu égoïste » décide de ne pas rendre la main et monopolise le processeur. Un bon exemple d'ordonnancement collaboratif n'est autre que les co-routines, des fonctionnalités de certains systèmes d'exploitations ou langages de programmation. Avec l'ordonnancement préemptif, c'est le système d'exploitation qui stoppe l'exécution d'un processus. L'ordonnancement préemptif se base souvent sur la technique du quantum de temps : chaque programme s'exécute durant un temps fixé une bonne fois pour toute. En clair, toutes les « x » millisecondes, le programme en cours d'exécution est interrompu et l'ordonnanceur exécuté. Ainsi, il est presque impossible qu'un processus un peu trop égoïste puisse monopoliser le processeur au dépend des autres processus. La durée du quantum de temps doit être grande devant le temps mis pour changer de programme. En même temps, on doit avoir un temps suffisamment petit pour le cas où un grand nombre de programmes s'exécutent simultanément. Généralement, un quantum de temps de 100 millisecondes donne de bons résultats. Divers algorithmes permettent de choisir un processus dans la file d'attente.
Pour faire simple, on peut classer ces algorithmes en trois classes : traitement par lots ; traitement interactif ; traitement temps réel. Nous n'aborderons pas le traitement temps réel, vu qu'il s'agit d'un cas assez spécial qui demanderait une petite connaissance de ce qu'est le temps réel et les contraintes associées.
Mais nous allons voir le traitement par lots et le traitement interactif. Traitement par lots Tout d'abord, on va commencer par un système d'exploitation qui exécute des programmes les uns après les autres, jusqu'à ce qu'ils finissent leur travail ou décident eux-mêmes de stopper leur exécution, soit pour accéder à un périphérique, soit pour laisser la place à un autre programme. Ce genre de chose est notamment très utilisé dans les banques ou les grosses entreprises, pour sauvegarder les comptes ou mettre à jour de grosses bases de données. Avec l'algorithme Premier arrivé, premier servi, l'ordonnanceur exécute le programme entré dans la file d'attente : les programmes sont exécutés dans l'ordre dans lequel ils sont rentrés dans la file d'attente. Cet algorithme est particulièrement trivial à programmer : une simple liste chainée suffit. Avec l'algorithme Dernier arrivé, premier servi, l'ordonnanceur exécute le programme entré en dernier dans la file d'attente et non le premier. Cet algorithme peut souffrir d'un phénomène assez particulier : dans certains cas, un programme peut très bien mettre énormément de temps avant d'être exécuté. Si vous exécutez beaucoup de programme, ceux-ci seront rentrés dans la file d'attente avant les tout premiers programmes. Ceux-ci doivent alors attendre la fin de l'exécution de tous les programmes rentrés avant eux. Avec l'algorithme du job le plus court en premier, on suppose que les temps d'exécution des différents programmes sont connus à l'avance et sont parfaitement bornés. Cette contrainte peut sembler absurde, mais elle a un sens dans certains cas assez rares dans lesquels on connait à l'avance le temps mis par un programme pour s'exécuter.
Dans ce cas, l'algorithme est simple : on exécute le programme qui met le moins de temps à s'exécuter en premier. Une fois celui-ci terminé, on le retire de la file d'attente et on recommence. L'algorithme Shortest Remaining Time Next est une variante préemptive de l'algorithme précédent.
Le fonctionnement est donc identique, à un point de détail prêt : que se passe-t-il si un nouveau processus est ajouté ? Dans ce cas, on compare le temps d'exécution du processus ajouté avec celui du processus en cours d'exécution. Si le processus ajouté prend moins de temps, alors il est exécuté directement. Néanmoins, cet algorithme a un léger problème : dans certaines conditions, des processus peuvent attendre indéfiniment leur tour.
En effet, il suffit pour cela d'ajouter en permanence de nouveaux processus plus prioritaires avant leur exécution, qui passeront systématiquement devant le processus en attente.
Traitement interactif Maintenant, il est temps de passer aux algorithmes que l'on trouve sur nos OS actuels. Il s'agit des algorithmes destinés à des environnements multimédias où l'utilisateur veut une réactivité optimale. Une première idée consiste à adapter l'algorithme du « job le plus court en premier » sur les systèmes interactifs. Le seul problème est que le temps d'exécution des processus n'est pas connu. La seule solution consiste à effectuer une estimation de ce temps d'exécution à partir des temps d'exécution précédents. L'algorithme du tourniquet utilise la méthode du quantum de temps. Avec cet algorithme, chaque programme est exécuté l'un après l'autre, dans l'ordre. Quand son quantum se termine, le programme doit attendre que tous les autres programmes aient eu droit à leur tour. Pour cela, à la fin de chaque quantum de temps, on place le programme à la fin de la liste d'attente. Cet algorithme est redoutablement efficace et des OS comme Windows ou Linux l'ont utilisé durant longtemps. Mais cet algorithme a un défaut : le choix du quantum de temps doit être calibré au mieux et n'être ni trop court, ni trop long. L'algorithme des files d'attentes multiple-niveau ordonnance aux mieux un ensemble de programmes aux temps d'exécution et aux particularités disparates en donnant la priorité aux programmes rapides et à ceux qui accèdent souvent aux périphériques. Cet algorithme utilise plusieurs files d'attentes, classées de la plus basse à la plus haute, auxquelles ont donne des quantum de temps différents. Plus une file est haute, plus son quantum est petit. Quand on exécute un programme, celui-ci est placé dans la file d'attente la plus haute. Si un programme utilise la totalité de son quantum de temps, c'est le signe qu'il a besoin d'un quantum plus gros : il descend alors d'un niveau. La seule façon pour lui de remonter d'un niveau est de ne plus utiliser complètement son quantum de temps.
Ainsi, un programme va changer de file suivant son temps d'exécution et se stabilisera dans la file dont le quantum de temps est optimal (ou presque). L'ordonnancement par loterie repose sur un ordonnancement aléatoire. Chaque processus se voit attribuer des tickets de loterie : pour chaque ticket, le processus a droit à un quantum de temps. À chaque fois que le processeur change de processus, il tire un ticket de loterie et le programme qui a ce ticket est alors élu. Évidemment, certains processus peuvent être prioritaires sur les autres : ils disposent alors de plus de tickets de loterie que les autres. Dans le même genre, des processus qui collaborent entre eux peuvent échanger des tickets. Comme quoi, il y a même moyen de tricher avec de l'aléatoire... Changements d'états Pour que notre programme puisse reprendre où il en était quand l'ordonnanceur lui redonne la main, il faut impérativement qu'un certain nombre de données soient restaurées. Par exemple, les registres du processeur doivent redevenir ce qu'ils étaient quand le programme a été interrompu. Le contexte d'exécution d'un processus est l'ensemble de ces informations.
Ce contexte d'exécution est sauvegardé en RAM quand l'ordonnanceur interrompt l'exécution du programme. Pour reprendre l'exécution d'un programme là ou il en était, il suffit de remettre en place le contexte sauvegardé. Une fois un processus élu, il doit être exécuté par notre processeur, c'est le rôle du dispatcher.
Celui-ci stoppe le programme en cours d'exécution et restaure le contexte du programme élu. Le système d'exploitation doit donc mémoriser les contextes de tous les processus. Cette ensemble de contexte est appelé la table des processus. Certains systèmes d'exploitation utilisent deux listes : une pour les processus prêts et une autre pour les processus bloqués. L'ensemble des processus prêts est ce qu'on appelle la file d'attente. Quand un programme démarre, celui-ci est ajouté à cette file d'attente. Les programmes ayant fini leur exécution ou qui passent en état bloqué sont retirés de la liste d'attente. Cette file d'attente est implémentée par ce qu'on appelle une liste chainée. Bien sûr, cette file d'attente n'accepte qu'un nombre limité de programmes : l'ordonnanceur décide toujours si oui ou non un programme peut être ajouté à cette liste et le nombre de programmes dans cette liste a une limite fixée par le système d'exploitation. Communication entre processus Dans les chapitres précédents, nous avons fait la confusion entre programmes et processus, notamment pour ce qui est de la protection mémoire.
Nous avons dit que deux programmes ne pouvaient pas se marcher l'un sur l'autre, dans le sens où chaque programme n'a accès qu'à une portion de mémoire qui lui est dédiée et pas au reste de la mémoire. Cela vaut toujours pour les processus, qui sont placés dans des espaces mémoire dédiés où seuls eux peuvent écrire, on parle d'isolation des processus. Il faut cependant noter que tous les systèmes d'exploitation n'implémentent pas cette isolation des processus, MS-DOS en étant un exemple. Avec isolation des processus Des processus peuvent avoir besoin de s'échanger des données alors qu'ils s'exécutent sur un même ordinateur, ce qui est très compliqué sur les systèmes avec isolation des processus. Sur ces systèmes, on est obligé d'utiliser des mécanismes de communication entre processus. La méthode la plus simple consiste à partager un bout de mémoire entre processus, qui pourront lire ou écrire dedans pour échanger des données (cela peut être un fichier, un morceau de mémoire RAM, etc). Cette méthode pose toutefois des problèmes assez spécifiques et plutôt complexe, aussi, nous ne les aborderons pas ici. Mais il existe des méthodes plus simples, que nous allons voir maintenant. Échange de messages L'échange de messages est un autre moyen pour faire communiquer deux processus. Avec celui-ci, les processus s'échangent des blocs de données de taille fixe, appelés messages. Ce message contient des données, qui peuvent être de tout type simple (entier positif ou négatif, flottant positif ou négatif, caractère isolé ou chaine de caractères) ou une combinaison de ces éléments (plusieurs nombres, des nombres et une chaine de caractères, ...). De plus, il contient un type, un entier positif, choisi par le programmeur, qui indique quel est le contenu du message. Il n'y a que trois restrictions sur le contenu du message : le type doit être présent ; le format des données doit être connu ; la taille des données doit être connue pour pouvoir être stockée par le processus qui recevra le message. Sur certaines implémentations, les messages sont envoyés directement, sans aucune mise en attente. Le processus destinataire du message est alors prévenu qu'on souhaite communiquer avec lui et doit arrêter ce qu'il fait pour échanger avec l'émetteur du message. Dit autrement, les deux processus doivent se synchroniser pour l'échange. On parle alors tout simplement de passage de message. Cela fonctionne bien quand les deux processus ne sont pas sur le même ordinateur et communiquent entre eux via un réseau local ou par Internet. Avec les tubes et les files de messages (message queue en anglais ou MSQ en abrégé), le processus destinataire d'un message et son émetteur n'ont pas besoin de se synchroniser.
Il est dès lors possible d'envoyer un message sans avoir à prévenir le processus destinataire. Pour cela, les messages que s'envoient les processus sont mis en attente dans une file d'attente (tant que celle-ci n'est pas remplie, auquel cas le processus émetteur se bloque). Les processus destinataires peuvent consulter cette file de message quand ils le souhaitent, notamment pour voir si on leur a envoyé des données (si la file d'attente est vide, ils se bloquent en attendant des données qui leur sont destinés). Dans tous les cas, les données envoyées via la file d'attente doivent de préférence être récupérées dans l'ordre d'envoi. Pour cela, le processus récupère systématiquement le message le plus ancien. Ainsi, les messages sont triés dans leur ordre d'envoi et sont retirés dans cet ordre par le processus destinataire. Notez que j'ai bien dit « retiré » et non pas « lus » : une fois qu'un message est consulté, il est retiré de la file. C'est ce qu'on appelle un fonctionnement « FIFO » pour First In First Out, premier entré, premier sorti.
Les tubes Le premier type de file d'attente est celui des tubes. La différence entre les deux tient dans le fait que les tubes permettent à seulement deux processus de communiquer, qui plus est dans un seul sens, là où une file de message ne souffre pas de ces limites. Il existe deux types de tubes : les tubes anonymes et les tubes nommés. Un tube anonyme n'est disponible que pour le processus qui l'a créé et ses fils.
De plus, il est détruit sitôt qu'il n'a plus d'utilité. Toute rupture de communication (perte du processus qui lit ou ce celui qui écrit) entraine dès lors la fermeture du tube. Un tube nommé est un tube qui permet la communication entre processus qui n'ont pas de lien de parenté. L'un d'eux écrit dans le fichier, l'autre lit ce fichier. Le tube reste en place une fois la communication terminée. Pour le reste, le fonctionnement est le même que pour un tube anonyme. Les files de messages Après avoir vu les tubes, il est temps de passer aux files de messages. On peut les voir comme un tube partagé entre plusieurs processus, chacun pouvant émettre à un ou plusieurs autres processus. De plus, chaque processus peut aussi bien lire qu'écrire dans la file de message. Créer une file de message se fait assez simplement, souvent avec un appel système. Toutefois, il faut pouvoir identifier et sélectionner celle-ci parmi toutes les autres MSQ qui peuvent avoir été créées. Pour cela, le système d'exploitation utilise un système de clé, c'est-à-dire un nombre choisi par le programmeur ou qui est généré automatiquement. Avec cette clé on génère une MSQ qui sera identifiée par un numéro dans le système. Une fois la MSQ en place, tout processus qui souhaite l'utiliser doit récupérer la clé de la file de messages. À partir de ce moment, les processus rattachés peuvent déposer ou récupérer un message. Pour récupérer un message, le processus se connecte à la MSQ et demande à récupérer les messages. Là encore, le processus récupère systématiquement le message le plus ancien. Contrairement à ce qui se passe avec le passage de message, la file d'attente est persistante. Elle reste donc en place une fois créé, et doit être supprimée soit par l'un des processus soit par une action des administrateurs système. Cette caractéristique permet de déposer des messages même lorsque le processus receveur est absent (comme dans votre boite mail ou postale : vous recevez toujours le courrier même si vous n'êtes pas chez vous). En revanche, cela peut mener à la saturation de la table. Il faudra donc toujours veiller à supprimer une file de messages lorsqu'elle n'est plus utilisée. Attention que si une MSQ est supprimée, les messages qu'elle contenait au moment de la suppression sont perdus, sans possibilité de les récupérer. Signaux Enfin, il existe une dernière méthode de communication inter-processus : les signaux. Lorsqu'un processus souhaite signaler un évènement à un autre processus, il lui envoie un numéro appelé un signal. La plupart concernent des erreurs, c'est-à-dire qu'ils sont envoyés à un processus lorsque celui-ci commet une « faute », une opération interdite, mais ce n'est pas systématique.
Il en existe plusieurs qui sont définis par le système d'exploitation. Le tableau d'exemple ci-dessous est celui des signaux du noyau Linux.
Numéro (dépend de l'implémentation ou du processeur) Nom Signification 1 SIGHUP Déconnexion du terminal ou mort du processus de contrôle 2 SIGINT Interruption depuis le clavier 3 SIGQUIT Quitter (depuis le clavier) 4 SIGILL Instruction illégale 5 SIGTRAP Un point d'arrêt a été trouvé 6 SIGABRT Signal d'arrêt depuis la fonction abort() du langage C 6 SIGIOT Synonyme de SIGABRT 7 SIGEMT Terminaison 8 SIGFPE Erreur mathématique en virgule flottante 9 SIGKILL Signal KILL. Permet de terminer un processus en urgence 10 SIGBUS Erreur de bus 11 SIGSEGV Référence mémoire invalide 11 SIGSEGV Référence mémoire invalide 12 SIGSYS Mauvais argument de routine 13 SIGPIPE Tentative d'écriture dans un tube sans lecteur à l'autre bout 14 SIGALARM Envoyé quand un timer atteint la durée décomptée 15 SIGTERM Terminaison normale de processus 16, 10, 30 SIGUSR1 Au choix de l'utilisateur 16 SIGSTKFLT Erreur de pile sur coprocesseur 16, 23, 21 SIGURG Un socket a le flag "urgent d'activé 17, 12, 31 SIGUSR2 Au choix de l'utilisateur 17, 20, 18 SIGCHLD Processus fils arrêté ou terminé 17, 19, 23 SIGSTOP Arrêt du processus 18, 20, 24 SIGTSTP Stop déclenché depuis un terminal 18 SIGTSTP Idem SIGCHLD 19, 18, 25 SIGCONT Continuer si le processus est arrêté 20, 21, 26 SIGTTIN Lecture depuis un terminal (en arrière-plan) 20, 21, 26 SIGTTOU Écriture depuis un terminal (en arrière-plan) 23, 29, 22 SIGIO Entrées-sorties à nouveau possibles - SIGPOLL Idem SIGIO 24, 24, 30 SIGXCPU Limite de temps CPU dépassée 25, 31 SIGFSZ Taille de fichier excessive 26, 28 SIGVTALRM Alarme virtuelle 27, 29 SIGPROP Profile Alarm Clock 28, 20 SIGWINCH Fenêtre redimensionnée 29, 30, 19 SIGPWR Chute d'alimentation 29 SIGINFO Idem SIGPWR - SIGLOST Perte de verrou de fichier 31 SIGUNUSED Signal inutilisé Un signal n'est pas toujours immédiatement transmis au processus concerné. En effet, celui-ci peut très bien être occupé et ne pas pouvoir le traiter directement. Il existe donc un système pour placer en attente ces signaux et les mettre à disposition du processus. Cette mise en attente prend la forme de drapeaux (flags), un par signal reçu. Si un signal a été reçu, on lève le drapeau correspondant à ce signal précis, sinon le drapeau reste baissé. Un signal qui est ainsi en attente et qui n'a pas encore été transmis est appelé un signal pendant. Lorsque le processus est à nouveau disponible, il reçoit les signaux qui étaient en attente (on dit qu'ils sont délivrés au processus concerné). Cette transmission a lieu lors d'un retour du processus dans le mode utilisateur. À partir de ce moment, le processus peut : ignorer le signal (même si certains signaux ne peuvent pas être ignorés, comme SIGKILL et SIGCHLD) ; traiter le signal avec l'action définie par le système ; traiter le signal de façon spécifique. Le programmeur du logiciel a la possibilité de traiter les signaux comme il l'entend, que ce soit pour ignorer le signal ou lancer une action spécifique. Dans le second cas, le programmeur peut prévoir une fonction pour gérer le signal reçu appelée un handler

(gestionnaire) de signal. Dans le cas où le programmeur n'a pas prévu d'action particulière pour un ou plusieurs signaux, le système d'exploitation a prévu une action par défaut, les plus courantes étant les suivantes : ignorer le signal ; la fermeture (abandon) du processus ; la fermeture du processus et l'écriture des causes dans un fichier spécifique appelé fichier core ; s'arrêter (se mettre en pause) ; reprendre (fin de la pause, au boulot !) . Sans isolation des processus Par convention, on qualifie de processus tout programme pour lequel il y a isolation des processus, les autres formes de programmes pouvant exister. Avec cette définition, un processus est simplement un fichier exécutable chargé en mémoire et isolé des autres processus/programmes.
Mais cette définition ne dit pas que ce processus correspond à un seul programme qui s'exécute d'un seul bloc. Il est en effet possible de rassembler plusieurs programmes dans un seul processus : ces sous-programmes sont appelés des threads. Les threads peuvent s'exécuter sur le processeur tout comme les processus : on peut ordonnancer des threads ou les exécuter en parallèle sur des processeurs séparés. Chaque thread possède son propre code à exécuter et sa propre pile d'appel.
Cela facilite beaucoup la programmation d'applications qui utilisent plusieurs processeurs. Cependant, les threads doivent partager leur zone de mémoire avec les autres threads du processus : ils peuvent ainsi partager des données sans devoir passer par de lourds systèmes de communication inter-processus. Comparaison entre un et plusieurs threads Le changement de contexte entre deux threads est beaucoup plus rapide que pour les processus. En effet, le fait que les threads se partagent la même mémoire évite certaines manipulations, obligatoires avec les processus. Par exemple, on n'est pas obligé de vider le contenu des mémoires caches sur certains processeurs. Généralement, les threads sont gérés en utilisant un ordonnancement préemptif, mais certains threads utilisent l'ordonnancement collaboratif (on parle alors de fibers). Threads utilisateurs Les threads utilisateurs sont des threads qui ne sont pas liés au système d'exploitation. Ceux-ci sont gérés à l'intérieur d'un processus, par une bibliothèque logicielle. Celle-ci s'occupe de la création et la suppression des threads, ainsi que de leur ordonnancement. Le système d'exploitation ne peut pas les ordonnancer et n'a donc pas besoin de mémoriser les informations des threads. Par contre, chaque thread doit se partager le temps alloué au processus lors de l'ordonnancement : c'est dans un quantum de temps que ces threads peuvent s'exécuter. Image adaptée d'une image de Silberschatz, Abraham provenant de wikicommons Threads noyaux Les threads noyaux sont gérés par le système d'exploitation, qui peut les créer, les détruire ou les ordonnancer. L'ordonnancement est donc plus efficace, vu que chaque thread est ordonnancé tel quel. Il est donc nécessaire de disposer d'une table des threads pour mémoriser les contextes d'exécution et les informations de chaque thread. Image adaptée d'une image de Silberschatz, Abraham provenant de wikicommons Allocation mémoire Sommaire