Système – DS 2010

# questions de cours

1. Le driver.
2. Sous forme de fichiers. On y accède comme à des fichiers.
3. La hiérarchie permet d’économiser de la mémoire en n’allouant que les tables dont on a besoin.
4. Le DMA permet de recopier des périphériques vers la mémoire sans occuper le processeur. La copie se fait donc en économisant cette ressource.
5. Sur l’hypothèse qu’une page anciennement utilisée a peu de chances d’être référencée dans un futur proche. (ou son corollaire)
6. Shortest Job Next : famine en cas d’arrivée constante de petites tâches.
7. Les pages de petite taille évitant d’avoir trop de mémoire non-utilisée (moins de « gachis » de mémoire). Puisque plus les pages sont grandes, plus il y a de chances d’avoir des utilisations partielles de ces pages. **Inconvénients** **:** petites pages => plus de pages => grosse table de pages.
8. Thread utilise un espace mémoire partagé alors que les processus ont chacun leur espace mémoire.
9. Eviter les défauts de page, et les recopies associées. On doit utiliser une mémoire virtuelle pour protéger les variables de chaque processus.
10. 232 / (8 \* 1024) = 524 288

# Remplacement de pages

## FIFO

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 5 | 5 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 0 | 0 | 0 | 0 | 0 |
|  | 0 | 0 | 0 | 0 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 1 | 1 |
|  |  | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 3 | 3 | 3 | 3 | 2 |

12 défauts de page. Attention : au premier, 5 n’est pas en mémoire, même si la mémoire est vide, c’est un défaut de page.

## LRU

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 5 | 5 | 2 | 2 | 2 | 4 | 4 | 3 | 3 | 3 | 3 | 1 | 1 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  |  | 1 | 1 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  | 4 | 0 | 0 | 1 |  |  |
|  |  |  |  |  | Correction : | | | 3 | 3 | 3 | 3 |  |  |
|  |  |  |  |  |  |  |  | 2 | 2 | 2 | 2 |  |  |

9 défauts de pages.

## LRU Horloge

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 5 | 5 | 2 | 2 | 2 | 2 | 2 | 4 | 4 | 4 | 4 | 3 | 3 | 3 |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 1 | 1 |
|  |  | 1 | 1 | 1 | 1 | 3 | 3 | 4 | 3 | 3 | 0 | 0 | 0 | 2 |

# Gestion de processus

Cf. ma correction

If(fork() != 0){

If(fork() != 0){

Fork() ;

}

Else{

If(fork() != 0){

Fork() ;

}

}

}

Else{

Fork() ;

}

# Section critique

Il n’aurait pas fallut faire une attente active car ici, si le verrou est LIBRE, plusieurs processus peuvent dépasser le while de la fonction et se placer à OCCUPE.

# Shifumi

## Masques

Void handler(int sig){

}

Main(){

Int ff[2]) ;

Pipe(ff)

Int fp[2] ;

Pipe(fp) ;

Struct sigaction act ;

Sig emptyset (&act.su.mask);

Act.su\_flags = 0;

Act.su\_handler = handler;

Signal(SIGUSR\_1, &act, NULL);

Sigset\_t \*ancienmask, \*nouveauMask;

Ancienmask = (sigset\_t \*)malloc(…) ;

Nouveaumask = (sigset\_t \*)malloc(…) ;

Sigemptyset(nouveaumask) ;

sigaddSet(nouveaumask, SIGUSR\_1) ;

sigaddset(nouveaumask, SIGUSR\_2) ;

pid\_t pid = fork() ;

if(pid != 0){ // Père

close(fp[0]);

close(ff[1]);

while(1){

string res = tire(pierre, papier, ciseaux);

kill(pid, SIGUSR\_1);

suspend(); //Attente signal SIGUSR2

write(fp[1], res) ;

read(ff[0], resAdversaire) ;

if(resAdversaire > res) // Adversaire gagne

kill(pid, SIGUSR\_1) ;

}

}

Close(fp[1]) ;

Close(ff[0]) ;

}

7) Il faut ajouter un handler et un signal SIGINT qui demande à chaque processus d’afficher le nombre de victoires.

Void handler\_affiche(int sig){

Printf(« score : %d », score) ;

}

8) Pour bloquer l’affichage du score tant que le round n’est pas fini, la solution serait de masquer les signaux pendant le round et de les démasquer à la fin.

# Salle de bain

Q9 ) **Q10)**

Init(mutex, 1) ;

Init(compteur, MAX) ;

Init(semG, 1) ;

Init(semF, 1) ;

Int etat ; // -1 garcon ; 0 libre ; 1 fille

Int nb = 0 ;

**Init(wait, 3) ; //Pour éviter que les personnes puissent être doublées**

Void boy\_enter(){

**P(wait) ;** //idem pour les filles

Bool ok = false;

P(semG);

While( !ok){

P(mutex) ;

ok = (etat == libre || etat == garcon);

If(etat == libre)

Etat = garcon;

V(mutex);

}

P(compteur);

P(mutex);

Nb++;

V(mutex);

V(semG);

**V(wait);**

}

Void boy\_leave(){

P(mutex);

Nb--;

If(nb == 0)

Etat = libre;

V(compteur);

V(mutex);

}

Q11) Il n’y a pas de famine

# Producteurs – consommateurs

Sem mutex[nbInfosMAX] //init à 1

Init(casesPleines, 0) ;

Init(casesVides, nbInfosMAX) ;

Bool infos[nbInfosMAX] ;

Int rebus ;

**Consommateur :**

P(casesPleines);

Int i = 0 ;

P(Mutex[i]);

While(infos[i] == 0){

V(mutex[i]);

i++ ;

P(mutex[i]);

}

If(consommable(infos[i])){

Info[i] = false;

V(mutex[i]);

}

Else{

Rebus.add(i);

V(mutex[i]);

V(casesVides);

}

**Consommateur particulier :**

If(rebus != 1){

P(Mutex[rebus[0]]);

Info[rebus[0]] = false;

V(Mutex[revus]);

Revus.enleve(0);

}

**Producteur :**

Bool aProduit = false;

While(!aProduit){

If(infos[i] == 0){

P(mutex[i]);

<product> // infos[i] = 1;

V(mutex[i]);

aProduit = 1;

}

Else{

I++;

If(I == nbInfosMAX)

I = 0;

}

}