Conservatoire **N**ational des **A**rts et **M**étiers **ANNEE UNIVERSITAIRE 2008-2009**	Centre d'enseignement de Marne La Vallée Bâtiment IFI 2 allée du Promontoire 93160 NOISY LE GRAND

EXAMEN DE : LINUX Principes et programmation (NSY103)
PROFESSEUR : Mr Cherrier
SESSION : Juin 2009 **DOCUMENTS AUTORISES :**
~~OUI~~ / NON.
DUREE DE L'EPREUVE : 2 heures

SUJET DE L'EPREUVE

cf page suivante

Processus(8 pts)

1. Qu'est ce qu'un processus ? (1pt)
2. Quels sont les différents états d'un processus ? (1pt)
3. Qu'est ce qui organise les processus dans un système multitâche ? Décrire son fonctionnement (2pts)
4. Définir la notion de temps réel (1pt)
5. Écrire un programme lanceur qui lance un processus fils qui exécutera le programme dont le nom a été passé en paramètre, dont la sortie sera redirigée dans le fichier (le nom de ce fichier a été passé en second paramètre). Les erreurs iront dans /dev/null : (3 pts)
   par exemple : si on tape « lanceur ls bilan » (ce programme lanceur crée un fils qui va exécuter ls, crée le fichier bilan qui contiendra le résultat du ls, et les erreurs iront dans /dev/null)
   Nota : si le fichier bilan existait déjà avant, il sera vidé, et si il n'existait pas, il sera crée.

La mémoire (7pts)

1. Dans le cas d'un système d'exploitation multi-tâche, pourquoi doit on gérer la mémoire ? (Décrire les besoins) (2pts)
2. Quel circuit est chargé de la gestion de la mémoire, et que permet il ? (1pt)
3. Quels sont les deux modes de découpage de la mémoire ? (1pt)
4. qu'est ce qu'un défaut de page ? (1pt)
5. Décrire en détail les différents algorithmes de gestion de l'allocation des pages dans le cadre de la mémoire virtuelle. (2pts)

Les tubes (5pts)

1. Décrire le principe de fonctionnement des tubes (1pt)
2. Comparez les tubes nommés et tubes anonymes. (1pt)
3. La commande unix suivante ***tr [a-z] [A-Z]*** transforme tout le flux de *stdin* en majuscule sur son *stdout* :
   par exemple ***who | tr [a-z] [A-Z]*** donne *SYLVAIN PTS/0 2009-06-10 13:44 (:0.0)*)
   On vous demande d'écrire un petit programme qui lance cette commande dans un processus fils et permet d'envoyer des informations à ce fils, et d'en récupérer le résultat (votre programme restera simple, le main() ne fera qu'envoyer CouCou à ce fils, en récupérera le résultat, puis l'affichera sur l'écran) (3pts)

Conservatoire **N**ational des **A**rts et **M**étiers **ANNEE UNIVERSITAIRE 2008-2009**	Centre d'enseignement de Marne La Vallée Bâtiment IFI 2 allée du Promontoire 93160 NOISY LE GRAND

EXAMEN DE : LINUX Principes et programmation (NSY103)
PROFESSEUR : Mr Cherrier
SESSION : Juin 2009 **DOCUMENTS AUTORISES :**
~~OUI~~ / NON.
DUREE DE L'EPREUVE : 2 heures

CORRIGE DE L'EPREUVE

cf page suivante

Processus(8 pts)

1. Qu'est ce qu'un processus ? (1pt)

Un processus est un ensemble d'octets (en langage machine) en cours d'exécution, en d'autres termes, c'est l'exécution d'un programme. C'est une instance en mémoire d'un programme.

2. Quels sont les différents états d'un processus ? (1pt)

La naissance d'un processus a lieu après l'appel système `fork` exécuté par un autre processus. Il devient au bout d'un certain temps "prêt à s'exécuter". Il passe alors dans l'état "exécuté en mode noyau" où il termine sa partie de l'appel système `fork`. Puis le processus termine l'appel système et passe dans l'état "exécuté en mode utilisateur".

Passé une certaine période de temps (le quantum, variable d'un système à l'autre), l'horloge peut *interrompre* le processeur. Le processus rentre alors en mode noyau, l'interruption est alors réalisée avec le processus en mode noyau.

Au retour de l'interruption, le processus peut être `préempté` (étant resté tout son quantum de temps sur le cpu), c'est à dire, il reste prêt à s'exécuter mais un autre processus est élu.

Si le processus fait un opération d'entrée sorties (disque, clavier, ram), il est alors mis en `attente`. Lorsque le système pourra lui donner l'information attendue, ce processus sera de nouveau eligible, et repartira dans la file d'attente « prét à s'exécuter".

3. Qu'est ce qui organise les processus dans un système multitâche ? Décrire son fonctionnement (2pts)

C'est le scheduler (l'ordonanceur des tâches) qui se charge de la getion des processus. Il dispose de la table des processus, et son rôle est de permettre à chacun des processus, selon leurs besoins et la disponibilité du système, de s'exécuter tour à tour. Pour chaque processus, on dispose du contexte d'execution (l'état dans lequel se trouvait le processeur au moment de la dernière interruption). Le scheduler arme un 'timer' (réglé sur le quantum de temps prédéfini dans le SE), timer qui interompera l'execution du processus. Le scheduler repositionne le processeur en mode utilisateur, ce qui permettra au timer d'interrompre réellement cette tâche (car il est lui de niveau système). LE processus peut cependant être interrompu plus tôt, notamment, si il execute un appel système pour obtenir une entrée sortie. Les accès au processeur, la politique d'attribution de l'attention du processeur est organisée sous forme de file d'attente, éventuellemetn avec des priorité d'accès différentes, et peut par exemple obéir à une politique de type touniquet (round robin)

4. Définir la notion de temps réel (1pt)

Les ordonnanceurs travaillant en temps réel prennent en compte des engagements de temps de réponse : dans ce cas, chaque tâche définit le temps et les contraintes qui la concerne. Le scheduler étudiera l'ensemble du système, et determinera si il est réalisable (c'est à dire, si l'ensemble des contraintes demandées par les processus ne conduit pas à des impossibilités). Si le système est faisable, il est alors lancé. Les processus sont alors interrompus selon les priorités demandées, et non selon l'utilisation d'un quantum de temps.

5. Écrire un programme lanceur qui lance un processus fils qui exécutera le programme dont le nom a été passé en paramètre, dont la sortie sera redirigée dans le fichier (le nom de ce fichier a été passé en second paramètre). Les erreurs iront dans /dev/null : (3 pts)
   par exemple : si on tape « lanceur ls bilan » (ce programme lanceur crée un fils qui va exécuter ls, crée le fichier bilan qui contiendra le résultat du ls, et les erreurs iront dans /dev/null)
   Nota : si le fichier bilan existait déjà avant, il sera vidé, et si il n'existait pas, il sera crée.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char **argv)
{
```

```
  int fd;
  int null_fd;

  if (argc != 3)
  {
    fprintf(stderr, "%s COMMANDE FICHIER\n", argv[0]);
    return EXIT_FAILURE;
  }
  if ((fd = open(argv[2],
          O_CREAT | O_WRONLY | O_TRUNC,
          S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
  {
    perror("open");
    return EXIT_FAILURE;
  }
  if (dup2(fd, 1) == -1)
  {
    perror("dup2");
    return EXIT_FAILURE;
  }
    if ((null_fd = open("/dev/null", O_WRONLY)) == -1)
    {
      perror("open");
      return EXIT_FAILURE;
    }
    if (dup2(null_fd, 2) == -1)
    {
      perror("dup2");
      return EXIT_FAILURE;
    }


  if (execlp(argv[1], argv[1], NULL) == -1)
  {
    perror("execlp");
    return EXIT_FAILURE;
  }
  return EXIT_SUCCESS;
}
```

La mémoire (7pts)

1. Dans le cas d'un système d'exploitation multi-tâche, pourquoi doit on gérer la mémoire ? (Décrire les besoins) (2pts)

Dès que l'on veut faire fonctionner plusieurs processus au même moment dans un ordinateur, se pose la problèmatique du partage des ressources. Ces ressources sont par exemple le processeur, la mémoire, les disques, le réseau, etc.. Concernant la mémoire, ressource limitée et cruciale, le système doit en garantir l'accès, et la confidentialité (éviter qu'un processus entre dans un espace qui ne lui appartient pas). De plus, les besoins en mémoire d'un processus sont variables dans le temps, il peut en demander, en libérer. Il faut donc gérer dynamiquement de l'allocation d'espace mémoire en garantissant la sécurité des accès, et son utilisation optimum.

2. Quel circuit est chargé de la gestion de la mémoire, et que permet il ? (1pt)

La MMU (Memory Management Unit) est chargé de gérer l'accès à la mémoire, en offrant un dispositif offrant au processeur un accès à la mémoire qui semble contigüe (les adresses mémoires sont modifiées afin de donner l'impression de continuité), ainsi qu'un dispositif déclenchant une erreur si un processeur tente d'accéder à une adresse hors de son espace attribué (la Segmentation FAult)

3. Quels sont les deux modes de découpage de la mémoire ? (1pt)

LA mémoire peut être paginée (c'est à dire découpée en page de taille fixe, par exemple 4 Ko). Cela permet une gestion simple de la mémoire, considérée alors comme une sorte de tableau, dont on affecte les cases au fur et à mesure des besoins. LE problème de cette solution est qu'elle ne respecte

pas les besoins des processus. L'autre méthode consiste à s'intéresser aux besoins réels des processus, et de créeer des découpages adaptés. Cependant, se pose alors le problème de l'utilisation optimale de l'espace mémoire, les différents segments pouvant ne pas bien s'intercaler.

4. qu'est ce qu'un défaut de page ? (1pt)

Afin d'augmenter les possiblitité d'utilisation du système, on peut simuler un espace mémoire supérieur à la taille réellemetn disponible, grâce à la mémoire virtuelle. Il s'agit alors de recopier sur disques des pages mémoires peu utilisées, afin d'offrir cette espace à un processus demandeur. Cependant, lorsque le processus d'originie veut retrouver ses données, il se produit un défaut de page (puisque cet espace a finalement été attribué à un autre processus)

5. Décrire en détail les différents algorithmes de gestion de l'allocation des pages dans le cadre de la mémoire virtuelle. (2pts)

Le remplacement optimal : on affecte les pages selon les besoins passés et futurs. Impossible à réaliser en réel, le fonctionnement des processus étant dynamique, il faudrait connaître l'avenir

LE fifo: p, L'algorithme le plus simple est Premier Entré Premier Sorti (First-In-First-Out).Quand une victime doit être sélectionnée c'est la page la plus ancienne qui est sélectionnée. Ce mécanisme rapide et simple à programmer n'est malheureusement pas très efficace.

LRU (Least Recently Used page). Nous utilisons ici le vieillissement d'une page et non plus l'ordre de création de la page. On fait le pari que les pages qui ont été récemment utilisées le seront dans un proche avenir, alors que les pages qui n'ont pas été utilisées depuis longtemps ne sont plus utiles. L'algorithme LRU est un bon algorithme mais il pose de nombreux problèmes d'implémentation.

L'algorithme de la deuxième chance :Un bit associé à chaque page est positionné à 1 à chaque fois qu'une page est utilisée par un processus. Avant de retirer une page de la mémoire, on va essayer de lui donner une deuxième chance. On utilise un algorithme FIFO plus la deuxième chance:

Les tubes (5pts)

1. Décrire le principe de fonctionnement des tubes (1pt)

Les tubes sont un mécanisme de communication qui permet de réaliser des communications entre processus sous forme d'un flot continu d'octets. Les tubes sont un des éléments de l'agrément d'utilisation d'UNIX. C'est ce mécanisme qui permet l'approche filtre de la conception sous UNIX.

2. Comparez les tubes nommés et tubes anonymes. (1pt)

Mécanisme de communication lié au système de gestion de fichier, les tubes nommés ou non sont des paires d'entrées de la table des fichiers ouverts, associées à une inode en mémoire gérée par un driver spécifique. Une entrée est utilisée par les processus qui écrivent dans le tube, une entrée pour les lecteurs du tube. Les tube nommés sont des tubes (`pipe`) qui existent dans le système de fichiers, et donc peuvent être ouverts grâce à une référence. Il faut préalablement créer le tube nommé dans le système de fichiers, grâce à la primitive `mknod` (`mkfifo`), avant de pouvoir l'ouvrir avec la primitive `open`.

3. La commande unix suivante ***tr [a-z] [A-Z]*** transforme tout le flux de *stdin* en majuscule sur son *stdout* : par exemple ***who | tr [a-z] [A-Z]*** donne *SYLVAIN PTS/0 2009-06-10 13:44 (:0.0)*)
   On vous demande d'écrire un petit programme qui lance cette commande dans un processus fils et permet d'envoyer des informations à ce fils, et d'en récupérer le résultat (votre programme restera simple, et ne fera qu'envoyer CouCou à ce fils, en récupérera le résultat, puis l'affichera sur l'écran) (3pts)

```
#include <ctype.h>
#include <unistd.h>


int main(int argc, char **argv) {
  pid_t pid;
```

```
  int pipealler[2];
  int piperetour[2];
  ssize_t bufsiz=10;
  char * buf[bufsiz];
  ssize_t readsiz;

  pipe(pipealler);
  pipe(piperetour);

  pid = fork();
  if (pid) {
    close(pipealler[0]);
    close(piperetour[1]);
  } else {
    dup2(pipealler[0],0);
    dup2(piperetour[1],1);
    close(pipealler[1]);
    close(piperetour[0]);
    execlp("tr","tr","[a-z]","[A-Z]",NULL);
    perror("exelp");
  }

  write(pipealler[1],"CouCou\n",8);
  close(pipealler[1]);
  readsiz=read(piperetour[0],buf,bufsiz);

  write(1,buf,readsiz);

  return EXIT_SUCCESS;
}
```