Table des matières

1 Création et terminaison de processus

1.1 Création de processus

— Un processus (le "parent") peut créer un autre (le "child")
— Un nouveau PCB est alloué et initialisé
— Exercice : exécuter 'ps auxwww' dans le shell ; PPID est le PID du parent

1.2 Héritage des attributs du processus enfant

— En POSIX, le processus enfant hérite de la plupart des attributs du parent
— UID, fichiers ouverts (doivent être fermés s'ils ne sont pas nécessaires ; pourquoi ?), cwd, etc.

1.3 Déplacement du PCB entre les files d'attente

— Pendant l'exécution, le PCB se déplace entre différentes files d'attente
— Selon le graphe de changement d'état
— Files d'attente : prêt, en attente/sommeil pour l'événement i (i=1,2,3...)

1.4 Processus zombie

— Après la mort d'un processus (exit() / interrompu), il devient un zombie
— Le parent utilise l'appel système wait* pour effacer le zombie du système (pourquoi ?)
— Famille d'appels système wait : wait, waitpid, waitid, wait3, wait4 ; exemple :
— pid_t wait4(pid_t, int *wstatus, int options, struct rusage *rusage) ;

1.5 Attente du parent pour le processus enfant

— Le parent peut attendre que son enfant termine ou s'exécute en parallèle
— wait*() bloquera sauf si WNOHANG est donné dans 'options'
— Exercice : lire 'man 2 wait'

2 Création et terminaison de processus

2.1 Création de processus

— Un processus (le "parent") peut créer un autre (le "child")
— Un nouveau PCB est alloué et initialisé
— Exercice : exécuter 'ps auxwww' dans le shell ; PPID est le PID du parent

2.2 Héritage des attributs du processus enfant

— En POSIX, le processus enfant hérite de la plupart des attributs du parent

— UID, fichiers ouverts (doivent être fermés s'ils ne sont pas nécessaires ; pourquoi ?), cwd, etc.

2.3 Déplacement du PCB entre les files d'attente

— Pendant l'exécution, le PCB se déplace entre différentes files d'attente
— Selon le graphe de changement d'état
— Files d'attente : prêt, en attente/sommeil pour l'événement i (i=1,2,3...)

2.4 Processus zombie

— Après la mort d'un processus (exit() / interrompu), il devient un zombie
— Le parent utilise l'appel système wait* pour effacer le zombie du système (pourquoi ?)
— Famille d'appels système wait : wait, waitpid, waitid, wait3, wait4 ; exemple :
— pid_t wait4(pid_t, int *wstatus, int options, struct rusage *rusage) ;

2.5 Attente du parent pour le processus enfant

— Le parent peut attendre que son enfant termine ou s'exécute en parallèle
— wait*() bloquera sauf si WNOHANG est donné dans 'options'
— Exercice : lire 'man 2 wait'

int main(int argc, char *argv[]) int pid = fork() ; if(pid==0) // // child // printf("parent=

Fiche Récapitulative

Génère une fiche récapitulative pour le chapitre suivant : Process creation & termination
Contenu du chapitre :

3 Création et terminaison de processus

3.1 Création de processus

— Un processus (le "parent") peut créer un autre (le "child")
— Un nouveau PCB est alloué et initialisé
— Exercice : exécuter 'ps auxwww' dans le shell ; PPID est le PID du parent

3.2 Héritage des attributs du processus enfant

— En POSIX, le processus enfant hérite de la plupart des attributs du parent
— UID, fichiers ouverts (doivent être fermés s'ils ne sont pas nécessaires ; pourquoi ?), cwd, etc.

3.3 Déplacement du PCB entre les files d'attente

— Pendant l'exécution, le PCB se déplace entre différentes files d'attente
— Selon le graphe de changement d'état
— Files d'attente : prêt, en attente/sommeil pour l'événement i (i=1,2,3...)

3.4 Processus zombie

— Après la mort d'un processus (exit() / interrompu), il devient un zombie
— Le parent utilise l'appel système wait* pour effacer le zombie du système (pourquoi ?)
— Famille d'appels système wait : wait, waitpid, waitid, wait3, wait4 ; exemple :
— pid_t wait4(pid_t, int *wstatus, int options, struct rusage *rusage) ;

3.5 Attente du parent pour le processus enfant

— Le parent peut attendre que son enfant termine ou s'exécute en parallèle
— wait*() bloquera sauf si WNOHANG est donné dans 'options'
— Exercice : lire 'man 2 wait'

4 Création et terminaison de processus

4.1 Création de processus

— Un processus (le "parent") peut créer un autre (le "child")
— Un nouveau PCB est alloué et initialisé
— Exercice : exécuter 'ps auxwww' dans le shell ; PPID est le PID du parent

4.2 Héritage des attributs du processus enfant

— En POSIX, le processus enfant hérite de la plupart des attributs du parent
— UID, fichiers ouverts (doivent être fermés s'ils ne sont pas nécessaires ; pourquoi ?), cwd, etc.

4.3 Déplacement du PCB entre les files d'attente

— Pendant l'exécution, le PCB se déplace entre différentes files d'attente
— Selon le graphe de changement d'état
— Files d'attente : prêt, en attente/sommeil pour l'événement i (i=1,2,3...)

4.4 Processus zombie

— Après la mort d'un processus (exit() / interrompu), il devient un zombie
— Le parent utilise l'appel système wait* pour effacer le zombie du système (pourquoi ?)
— Famille d'appels système wait : wait, waitpid, waitid, wait3, wait4 ; exemple :
— pid_t wait4(pid_t, int *wstatus, int options, struct rusage *rusage) ;

4.5 Attente du parent pour le processus enfant

— Le parent peut attendre que son enfant termine ou s'exécute en parallèle
— wait*() bloquera sauf si WNOHANG est donné dans 'options'
— Exercice : lire 'man 2 wait'

int main(int argc, char *argv[]) int pid = fork() ; if(pid==0) // // child // printf("parent=

La fiche doit inclure : 1. Les points clés 2. Les formules importantes 3. Les concepts essentiels 4. Les applications pratiques

Format : Utilise le même format que les autres blocs avec tcolorbox.