Master1 – POSIX 2005-2006 Contrôle Continu	- Durée : 2 heures - Toute documentation autorisée - Barème indicatif *Luciana Arantes, Bertil Folliot et Olivier Marin*

1. GESTIONS DES PROCESSUS (9 POINTS)

Considérez le code de la fonction **int nfork (int N, pid_t* pt_pid)** donné ci-dessous :

```
int nfork( int N, pid_t* pt_pid) {
  int i;
  *pt_pid =0;
   for (i=0; i<N && ((*pt_pid = fork())==0); i++);
   if (*pt_pid  == -1)
        return -1;
   else
        return i;
}
```

1.1

Expliquez en détail ce que fait la fonction ***nfork.***

En utilisant la fonction ***nfork,*** le programme ci-dessous crée une arborescence de processus.

```
1 : int main (int argc, char * argv[]) {
2 :   int N=3; int p=0;
3:    pid_t pid;
4:  do {
5:        N=N-p;
6:        if ((p=nfork (N,&pid)) == -1)
7:           exit (-1);
8:        else
9:          if (p==0)
10:            printf ("N=%d \n",N);
11:  } while (p!=0);
12:  if (N==p)
13:    printf ("p=%d\n",p);
14:  return 0;
   }
```

1.2

En considérant qu'aucune création de processus n'échoue, indiquez le nombre de processus créés ? Dessinez l'arborescence des processus en montrant les affichages de chaque processus.

1.3

Au lieu d'appeler la fonction *printf* dans la ligne 10, nous voulons appeler la commande ***echo,*** qui se trouve dans le répertoire */bin*. Remplacez la ligne 10 par l'appel de la commande ***echo*** en utilisant *execv*.

Le nombre de processus créés change-t-il ? Et l'affichage ?

1.4

Modifiez le programme original pour que le processus initial attende la fin de *tous* les processus créés.

2. SYNCHRONISATION DE PROCESSUS – SIGNAUX (6 POINTS)

2.1

Nous avons un processus P1 qui crée un processus P2 (fils de P1) qui à son tour crée un processus P3 (fils de P2 et petit-fils de P1). Lorsque le processus P3 est créé, il envoie un signal à son grand-père, processus P1, pour lui signaler sa création. Le processus P3 se termine juste après. Quand son père, le processus P2, prend connaissance de la terminaison de P3, il envoie un signal à P1, son père, pour signaler la terminaison de son fils. Ensuite, P2 se termine lui aussi.

Le processus P1 doit traiter les événements dans l'ordre décrit ci-dessus. Autrement dit, il doit premièrement traiter la délivrance du signal de P3 en affichant le message « *Processus P3 créé* », après la délivrance du signal de P2 en affichant « *Processus P3 terminé* » et à la fin afficher « *Processus P2 terminé* » lorsqu'il prend connaissance de la mort de son fils.

Programmez cette synchronisation.

3. ENTRÉES/SORTIES (5 POINTS)

On cherche à programmer une version concurrente de la commande 'grep', qui est appelée de la manière suivante :

```
$mygrep <expression> <repertoire>
```

L'utilisateur fournit un `repertoire` de recherche dans lequel chacun des fichiers réguliers sera entièrement lu afin de trouver les lignes contenant l'`expression` recherchée. Le programme affiche chacune des lignes trouvées sur la sortie standard.

3.1.

On s'intéresse d'abord à la fonction correspondant à la lecture d'un fichier pour y trouver les lignes contenant l'expression recherchée. Dans ce cas précis, pourquoi est-il plus intéressant d'utiliser la fonction `fgets` que n'importe quelle autre fonction de lecture ?

Donnez le code de la fonction qui effectue cette tâche et qui a pour signature :

```
void extract_from_file(char *parse_expr, char *src_filename)
```

On se servira de la fonction `char* strstr(char *s, char *s1)`, qui renvoie l'adresse de `s` si `s1` y est trouvée, `NULL` sinon. NB : une ligne a une taille maximale définie par LINE_MAX.

3.2.

Écrivez maintenant le code de la fonction permettant de parcourir un répertoire pour y trouver les fichiers réguliers et leur appliquer la fonction de recherche d'expression de la question précédente :

```
int browse_directory(char *parse_expr, char *dir_path)
```

Cette fonction crée autant de processus fils qu'il n'y a de fichiers réguliers dans le répertoire et renvoie le nombre de fils créés. Chacun des fils est chargé de lire un fichier pour vérifier si l'expression cherchée s'y trouve.

La fonction `browse_directory` sera appelée depuis le code suivant :

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

    if (argc != 3) {
        fprintf(stderrr, "usage: mygrep <expression> <directory>\n");
        exit(1);
    }
    printf("**** Searching for '%s' in %s ****\n", argv[1], argv[2]);
    browse_directory(argv[1], argv[2]);
    return 0;
}
```

NB : La structure `struct dirent` contient un champ entier `d_type`.

La fonction `mode_t DTTOIF (int dtype)` renvoie la valeur de `st_mode` correspondante.