Université de Rouen
Licence informatique
Module : Système I
2003-2004

TP 1 : Simulation

Pour chacune des questions qui suivent, produire un fichier `.c` (source du programme écrit en langage C), un fichier `.h` (types et entêtes des fonctions exportables), et un autre fichier `.c` contenant la fonction `main`. L'ensemble des exécutables produits lors de cette séance de travaux pratiques doit être fabriqué par un unique fichier `makefile` mis à jour à chaque question.

Vous devez écrire un fichier `.h` de façon à ce qu'il soit compilé une seule fois, quelque soit le nombre de fois où il est inclu. Vous pouvez vous assurer de cela en définissant une variable du processeur au début de chaque fichier `.h`, si la variable est déjà définie le contenu du fichier ne doit pas être recompilé (utiliser pour cela les directives du préprocesseur `#ifndef` ...`#endif`, et `#define`).

Rappelons qu'un fichier `makefile` décrit une suite d'entrées de la forme :

```
cible : source1 source2 ... sourcen
<tab>   action
```

Ce qui veut dire que `cible` dépend des fichiers source `source1`, `source2`, ...`sourcen`, et que si un des fichiers source est plus récent que `cible` (dans le cas où cible est un fichier) alors `action` doit être exécutée.

Par exemple, pour faire en sorte que le fichier objet `prog.o` doit être mis à jour à partir de deux fichiers `prog.c` et `prog.h`, on peut définir l'entrée du `makefile` comme suit :

```
prog.o : prog.c prog.h
<tab>   gcc -c prog.c
```

Question 1.- (1) Ecrire une fonction

```
void *allouerTab(size_t nelts, size_t size)
```

qui alloue un tableau de `nelts` éléments, chacun d'eux représentant `size` octets, et renvoie un pointeur vers la mémoire allouée. Dans le cas ou la mémoire est insuffisante, afficher un message d'erreur sur la sortie standard des erreurs (`stderr`) et arrêter le programme avec une valeur de retour indiquant l'echec (`EXIT_FAILURE`).

(2) Ecrire une fonction
`void reallouerTab(void *ptr, size_t nelts, size_t size)`
qui réalloue le tableau pointé par `ptr` avec une nouvelle taille `nelts`. Même réaction que `allouerTab` en cas d'echec.

(3) Ecrire une fonction
`void **allouerMat(size_t nlignes, size_t ncolonnes, size_t size)`
qui alloue une matrice de `nlignes` lignes et `ncolonnes` colonnes. Même réaction que `allouerTab` en cas d'echec.

(4) Ecrire une fonction
`void **reallouerMat(void **ptr, size_t nlignes, size_t ncolonnes, size_t size)`
qui réalloue la matrice pointée par `ptr` avec une nouvelle taille (`nlignes`, `ncolonnes`).

(5) Ecrire une fonction
`libererMat(void **ptr, size_t nlignes)`
qui libère toute la mémoire occupée par la matrice.

Question 2.- La fonction `int rand(void)` retourne à chaque appel un nombre entier pseudo-aléatoire compris entre 0 et `RAND_MAX`. Par défaut `rand` retourne toujours la même suite de nombres. Pour modifier la suite on utilise la fonction `void srand (unsigned int seed)` qui permet à `rand` de générer une suite différente suivant la valeur `seed` transmise à `srand`. Pour transmettre à `srand` une valeur `seed` différente à chaque appel on peut utiliser le temps (qui n'est pas le même d'un instant à l'autre comme vous le savez). On peut utiliser pour mesure du temps la fonction `time_t time(time_t *t)` qui retourne le nombre de secondes écoulées depuis le premier janvier 1970 à 00h 00m 00s GMT.

Ecrire une fonction `int newRandom(int nmin, int nmax)` qui retourne un entier pseudo-aléatoire compris entre `nmin` et `nmax`, et une fonction `void initRandom(void)` qui initialise la suite de nombres en utilisant le temps.

Question 3.- On veut simuler la création et la terminaison de processus de différents utilisateurs sur un système. On suppose pour simplifier que :

(1) le nombre d'utilisateurs logés est fixe (20 par défaut) ;

(2) à chaque instant peut se produire un événement de type création ou destruction d'un processus appartenant à un utilisateur quelconque ;

(3) en moyenne il se produit `N` événements (5 par défaut) dans le système et `Nmax` au maximum (10 par défaut).

Dans cette simulation on s'intéresse seulement au nombre de processus appartenant à chaque utilisateur à un moment donné. On stocke cette information dans une matrice dans laquelle chaque ligne représente le nombre de processus de chaque utilisateur à un moment donné. Une nouvelle ligne est enregistrée chaque seconde. L'enregistrement des informations s'arrête à la demande de l'utilisateur. A ce moment là toutes les informations enregistrées dans la matrice doivent être sauvées dans un fichier.