NOM :
Prénom :

Rattrapage de INF104, 12/05/2016

1h30 – Sans documents

Nous avons laissé de la place sur l'énoncé pour vos réponses. Vous pouvez donc rendre l'énoncé complété. Attention, ***mettez votre nom sur toutes les feuilles rendues.***

Partie 1 : Langage C et chaine de production (8 points)

Bases du langage C (2 points)

Question 1 (1 point) : On considère le code suivant :

```
int a = 5;

int * ptr = ____ ;

int b = ____;
```

Complétez ce code pour initialiser `ptr` avec l'adresse de la variable `a` ; et initialiser `b` avec la valeur pointée par `ptr`.

Question 2 (0,5 point) : Décrivez le fonctionnement de l'opérateur sizeof en C

Question 3 (0,5 point) : Considérons un pointeur `ptr` et un entier `k`. Ces deux notations sont elles équivalentes : `*(ptr+k)` et `ptr[k]` ? Justifiez votre réponse

NOM :
Prénom :

Chaine de production (2 points)

Question 4 (1 point) : On souhaite compiler séparément les fichiers `main.c` et `math_functions.c`, puis faire l'édition de lien entre les fichiers binaires résultants de cette compilation. Ecrivez les trois lignes de commandes qu'il faudra exécuter successivement pour réaliser cela.

Question 5 (1 point) : Quelles sont les différentes sections présentes dans un fichier binaire exécutable ?

Exercice langage C (4 points)

Question 6 (3 points) : Ecrire le code C qui implémente la fonction **strcpy**. Pour information, la fonction **`strcpy`** copie la chaine de caractères `src` dans la chaine de caractère `dst` (y compris le caractère de fin de chaine : '\0').

Indication : il ne faut pas faire appel à la fonction `malloc` dans ce code; on suppose que l'espace mémoire nécessaire pour stocker la chaine de caractère à partir de l'adresse `dst` a déjà été alloué.

Il faut donc que vous complétiez le code ci dessous:

```
void strcpy(char *dst, const char *src)
{
  // compléter le programme avec les informations ci-dessus.









}
```

Question 7 (1 point) : expliquer pourquoi l'usage de la fonction `malloc` dans l'implémentation de `strcpy` ci-dessous abouti à la création d'une fuite mémoire.

NOM :
Prénom :

Partie 2 : Systèmes d'exploitation (12 points)

Questions sur les processus (3 points)

On dispose sur un serveur de 3 vidéos correspondant à l'enregistrement de phases d'entrainements d'une équipe de football en préparation pour l'euro 2016. Chaque vidéo a été tournée à des dates différentes. On souhaite visualiser ces vidéos en parallèle. Pendant la visualisation, des algorithmes de traitement d'image sont utilisés pour analyser la performance des sportifs. Le but de notre programme est de paralléliser l'analyse des flux vidéo correspondant à chaque entrainement.

Question 8 (3 points) : En utilisant le canevas fournis ci-dessous, écrire le code C de la fonction main qui permet de traiter les 3 flux vidéo dans 3 processus différents (permettant ainsi leurs exécution en parallèle sur une machine multi-coeur). On suppose qu'on dispose d'une fonction `traiter_flux` qui prend en paramètre l'identifiant du flux sous la forme d'une chaine de caractères (on les identifiera ainsi : "flux1", "flux2", ou "flux3"). On suppose que `traiter_flux` traite l'intégralité du flux vidéo (pas besoin d'utiliser une boucle `while` dans le code que vous proposez).

```
#include <unistd.h>

// fonction à invoquer pour traiter un flux à partir de son identifiant.
void traiter_flux(char * id_flux) ;

int main(int argc, char* argv[])
{
  // compléter le programme avec les informations ci-dessus.









}
```

NOM :
Prénom :

Questions sur les signaux (5 points)

Dans cet exercice, nous nous intéressons au code du réveil matin qui vous permet d'arriver à l'heure en cours. Ce réveil possède trois fonctionnalités : 1 – sonner à l'heure de réveil de votre choix, 2 – arrêter de sonner pendant une courte durée, 3 – arrêter de sonner jusqu'à la prochaine heure de réveil. Le code fourni en annexe est un début de code, **à compléter,** pour représenter le fonctionnement du réveil. Pour le compléter, nous utiliserons les signaux suivants :

- SIGALRM, pour représenter l'arrivée de l'heure de réveil (déclencher la sonnerie).
- SIGUSR1, pour stopper temporairement la sonnerie, pendant un court laps de temps. Cette fonctionnalité est appelée « snooze » dans la description du code fourni.
- SIGUSR2, pour stopper la sonnerie jusqu'à la prochaine date de réveil. Cette fonctionnalité est appelée « stop » dans la description du code fourni.

Les indications sur le code fourni, ainsi que les lignes à compléter, sont décrites en commentaire dans le code. Notez que l'espace laissé pour compléter n'est pas indicatif du nombre de lignes à écrire.

Question 9 : Les questions qui suivent vont vous guider pour compléter le code à trous fourni en annexe. Il est attendu que vous n'utilisiez que des fonctions déclarées dans le code ou dans les APIs système que vous connaissez pour gérer les signaux. Vous pouvez répondre directement sur l'annexe fournie.

Question 9.1 (1 point) : complétez le code de la fonction `main` en suivant les indications données en commentaire. Au lancement du programme, il faut ignorer les signaux correspondants aux fonctionnalités snooze et stop : ces deux fonctions ne sont utiles que lorsque le réveil sonne. Enfin, il faut exécuter la fonction « on_alarm » lorsque le réveil sonne.

Question 9.2 (1,5 point) : complétez le code de la fonction « on_alarm ». Lorsque cette fonction est exécutée, cela signifie que le réveil sonne. On pourra donc l'arrêter ou demander à ce que le réveil sonne après un petit délai (fonctionnalité snooze). Tant que le réveil sonne, l'utilisateur peut utiliser cette fonctionnalité snooze. Complétez le code de la boucle while en conséquence.

Question 9.3 (1,5 point) : complétez le code de la fonction `on_stop` de sorte que le réveil sonne de nouveau après un délai obtenu en appelant la fonction `get_next_alarm_time`.

Question 10 (1 point) : quelle commande peut-on utiliser pour simuler le comportement d'un utilisateur qui active la fonctionnalité d'arrêt temporaire de la sonnerie (fonctionnalité snooze) ou d'arrêt de la sonnerie jusqu'à la prochaine date de réveil (fonctionnalité stop) ? On suppose que le signal SIGUSR1 et SIGUSR2 portent respectivement les numéro 30 et 31, et que la trace d'exécution commence par :

```
PID: 3065
RINGING !!!!!!
```

NOM :
Prénom :

Questions sur la pagination de la mémoire (4 points)

Question 11 (1 point) : Donner la définition d'un défaut de page.

Question 12 (1 point) : Donner la définition de la politique LRU utilisé en pagination mémoire.

Question 13 : On considère un processus nécessitant l'utilisation de 7 pages en mémoire. Le tableau ci-dessous donne la correspondance entre pages logiques et pages physiques.

Page logique	Page physique
0	4
1	5
2	1
3	9
4	7
5	6
6	2

Question 13.a (1 point) : En supposant que la taille d'une page est de 1 Ko (1024 octets; 1024 = 2^{10}), quelle est l'adresse physique correspondant à l'adresse logique 0x05F2 (format hexadecimal).

Donnez votre réponse au format hexadécimal.

Question 12.b (1 points) : En supposant que la taille d'une page est de 0.5 Ko (512 octets; 512 = 2^{9}), quelle est l'adresse logique correspondant à l'adresse physique 0x072E (format hexadecimal).

Donnez votre réponse au format hexadécimal.

Annexe à rendre
Nom, Prénom :

Extrait de code pour répondre à la question 9

```
// the alarm clock starts ringing when this function is called void
start_ringing();
// the alarm clock stops ringing when this function is called
void stop_ringing();
// returns the amount of time to reach the next alarm time
int get_next_alarm_time();

// function to execute when an alarm time is reached
void on_alarm(int num_sig);
// function to execute when a user temporarily stops the alarm
void on_snooze(int num_sig);
// function to execute when a user stops the alarm until the next alarm time
void on_stop(int num_sig);

// Boolean values to represent the state of the alarm clock (snoozing, stopped,
ringing)
char snooze_pushed=0;
char stop_pushed=0;
char alarm_ringing=0;

int main()
{
  printf("PID: %d\n", getpid());
  // To be completed
  // 1 - Ignore the signal snoozing the alarm, should not trigger
  // anything when the alarm is not ringing




  // 2 - Ignore the signal for stopping the alarm,
  // should not trigger anything when the alarm is not ringing




  // 3 - When alarm is triggered, execute the on_alarm function



  int duration = get_next_alarm_time();
  alarm(duration);
  while(1); // infinite loop: the alarm always work to be on time at school!
}

void on_snooze(int num_sig)
{
  snooze_pushed = 1;
}
```

Annexe à rendre

Nom, Prénom :

```
void on_stop(int num_sig)
{
  alarm_ringing = 0;
  // To be completed: make sure the alarm will ring for the next alarm time!




}

void on_alarm(int num_sig)
{
  start_ringing();
  alarm_ringing = 1;
  // To be completed: now that the alarm is ringing, stop and snooze
  // functions can be triggered



  while(alarm_ringing)
  {
    if(snooze_pushed)
    {
      // To be completed: ignore signals for snooze and stop while snooze
      // is being processed



      snooze_pushed=0;
      stop_ringing();
      sleep(2); // snooze time is predefined to 2 (seconds) for simulation;
      // sleep(N) blocks the process during N seconds.
      start_ringing();
      // To be completed: alarm is ringing again: snooze and stop function can be
      // reactivated




    }
  }
  stop_ringing();
}

int get_next_alarm_time()
{
  return 4; // next alarm time is predefined to 4 (seconds) for simulation;
}

void start_ringing()
{
  printf("RINGING !!!!!!\n");
}
```