Examen de SELC/INF104

1h30 – Sans documents

Correction en rouge dans le texte

Questions de C (8 points)

Question 1 (0 ,5 points) : Quelle est la taille, en mémoire et en octet, d'une variable de type char ?

1 octet

Question 2 (0,5 points) : on déclare une variable v, de type tableau de 3 entiers (int). Donner deux notations en C qui permettent d'accéder au dernier élément de ce tableau.

v[2] ou *(v+2)

Question 3 (0,5 points) : énumérer les zones mémoires principales utilisées par l'exécution d'un programme.

Pile, tas, statique (code/données)

Question 4 (0,5 points) : En considérant le programme suivant :

```
#include <stdio.h>
int main(int argc, char* argv[]){
 int J=0;
 if(J=1)
   printf("J vaut 1\n");
 else
   printf("Non, J vaut 0, J0!!!\n");
}
```

Le programme est-il compilable ? Quelle sera la trace d'exécution de ce programme ?

Compilable; J vaut 1

Question 5 (4 points) : compléter le code ci-dessous en suivant les consignes placées après ce code.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct competitor
{
  char * name[80];
  int time;
}competitor_t;

int nb_comp=0;

competitor_t * get_qualified_competitors(competitor_t * c, int
limit);

int main(int argc, char* argv[]){
 competitor_t tab[10];
 int ret=0;
 FILE * f = fopen("race_list.txt", "r");
 while(ret!=EOF)
 {
```

```
      ret = fscanf(f, "%s %d", tab[nb_comp].name,
                               &tab[nb_comp].grade);
      nb_comp++;
    }
    int nb_qualif=0;
    competitor_t * qualif = get_qualified_competitors(tab,
                                                      & nb_qualif,
                                                      90);
    for(i=0;i<nb_qualif;i++)
      printf("Qualified nb %d: %s \n", i+1, qualif[i].name)
    …
   }
```

```
extern int nb_comp;
competitor_t * get_qualified_competitors(competitor_t * c, int * nb, int time)
{
  int i = 0;
  competitor_t * res = (competitor_t*) malloc(nb_comp*sizeof(competitor_t));
  for(i=0;i<nb_comp;i++)
  {
    if(tab[i].time < time)
    {
      res[nb].name = c.name;
      res[nb].time = c.time;
      *nb=*nb+1;
    }
  }
  return res;
}
```

Ecrire l'implémentation de la fonction :
`competitor_t * get_qualified_competitors(competitor_t * c, int * nb, int time).`
Cette fonction copie, à partir du tableau c passé en paramètre, tous les compétiteurs qualifiés dans un nouveau tableau. Ce nouveau tableau est retourné par la fonction. Les compétiteurs qualifiés sont ceux pour lesquels le champ *time* de la structure est strictement inférieur à une limite passée en paramètre (paramètre *limit*) de la fonction (90 dans le code ci-dessus). Le nombre de compétiteurs qualifiés est mis à jour grâce au paramètre nb.

Question 6 (2 points) : Complétez la trace d'exécution obtenue par l'exécution du programme suivant.

```
printf("Taille de short: %d\n",sizeof(short));
char Tab[4] = {0,1,2,3};
short * Ptr1;
char * Ptr2;
Ptr1 = (short *)Tab;
Ptr2 = (char *)Tab;
printf("Representation hexadecimal de Ptr1: %04x\n", *Ptr1);
printf("Representation hexadecimal de Ptr2: %04x\n", *Ptr2);
```

Trace partielle, dans laquelle il est demandé de remplacer … par la valeur affichée:

```
Taille de short: 2
Representation hexadecimal de Ptr1: 0001
Representation hexadecimal de Ptr2: 0000
```

Questions chaines de compilation(4 points)

Question 7 (1 point): dans le programme donné à la question 5, à quoi correspond la ligne #include <stdio.h> ? quel est son effet lors de l'invocation de gcc pour compiler le programme?

#include <stdio.h> est une directive au préprocesseur qui permet d'insérer la déclaration des types et fonctions de la librairie stdio. Lors de l'appel à gcc, cette directive est remplacée par le contenu du fichier.

Question 8 (2 points): on souhaite écrire un makefile qui compile un programme C à partir de trois fichiers C : main.c, graph.c, path_computation.c. Compléter le makefile suivant en remplaçant les pointillés par le code qui convient.

```
#########################################
CC       = gcc
CFLAGS   = -c
LD       = gcc
OFILES   = main.o graph.o path_computation.o
#########################################

all: prog

#production de l'objet main.o (a completer)
main.o: main.c
        $(CC) $(CFLAGS) main.c

# production de l'objet graph.o (a completer)
graph.o:
        $(CC) $(CFLAGS) graph.c

# production de l'objet path_computation.o (a completer)
path_computation.o:
        $(CC) $(CFLAGS) path_computation.c


# production de l'executable prog (a completer)
prog: $(OFILES)
        $(LD) $(OFILES) -o prog
```

Question 9 (1 point): on suppose que la commande "ls" renvoie les informations suivantes, qui incluent les dates de dernières modifications des différents fichiers :

```
graph.c             27/01/2014 - 10:30
graph.o             27/01/2014 - 10:35
main.c              27/01/2014 - 11:25
main.o              27/01/2014 - 10:36
path_computation.c  27/01/2014 - 09:59
path_computation.o  27/01/2014 - 10:07
prog           27/01/2014 - 10:37
```

Que se passera-t-il, dans ces conditions, lors de l'invocation de make ?

main.c est recompilé (0,5 point). L'édition de lien est ré-exécutée (0,5 points).

Questions sur les processus (4 points)

On dispose sur un serveur de 3 vidéos correspondant à l'enregistrement des entrainements d'un skieur en préparation pour les jeux olympiques. On souhaite visualiser ces vidéos en parallèle tout en les synchronisant. Pendant la visualisation, des algorithmes de traitement d'image sont utilisés pour analyser la performance du sportif. L'objectif de la parallélisations du traitement des différentes vidéos est d'utiliser les temps pendant lesquels le traitement d'une

vidéo est bloqué en attente de nouvelles informations du serveur distant. Nous commençons par nous intéresser au problème de la parallélisations.

Question 10 (3 points) : En utilisant le canevas fournis ci-dessous, écrire le code C de la fonction main qui permet de traiter en parallèle les 3 flux vidéo. On suppose qu'on dispose d'une fonction traiter_vidéo qui prend en paramètre l'identifiant du flux sous la forme d'une chaine de charactères ("flux1", "flux2", ou "flux3"). On suppose que traiter_vidéo traite l'intégralité du flux vidéo (pas besoin d'utiliser une boulce while ici).

```
#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[])
{
  …
}
```

```
int f1, f2 ;
f1=fork() ;
if(f1==0)
  traiter_flux("flux1");
else if(f1>0)
{
  f2=fork();
  if(f2==0)
    traiter_flux("flux2");
  else if(f2>0)
    traiter_flux("flux3");
  else
    printf("error…
}
else
{
   printf("error…
}
```

Question 11 (1 point) : En cas de problème dans l'exécution d'un des traitements, on veut récupérer un identifiant de l'erreur. Indiquez quelle fonction C, sous UNIX, permet de récupérer ces informations dans la solution mise en œuvre ici. Préciser la signature de son (ou ses) paramètre(s)

Il faut utiliser la fonction wait(int*)

Questions sur la synchronization (4 points)

Question 12 (4 points) : On va maintenant synchroniser les vidéos, de telle sorte à traiter les images 4 par 4 dans chaque flux vidéo. On suppose qu'on dispose d'une fonction traiter_quatre_images, qui renvoie 0 lorsque la fin de la

vidéo a été atteinte. En utilisant des sémaphores dont vous choisirez la valeur initiale, représenter le pseudo code de synchronisation des 3 processus correspondant au traitement de chaque flux vidéo.
Format attendu de la réponse :

1) Définition et Initialisation des sémaphores (en utilisant la fonction INIT) et/ou variables Partagées

2) Utilisation des sémaphores (en utilisant les fonctions P et V) et/ou variables partagées :

```
Processus flux i (i=1..3)

traiter_flux(char*id_flux)
{
  while(c !=0)
  {
    // Ajouter synchronisation ici
    c = traiter_quatre_images(flux) ;
  }
}
```

Barrière

```
int n=0;
Init(SX,1);
Init(Barrier, 0);
#define NB_PROCESS 3
traiter_flux(char*id_flux)
{
  int i=0 ;
  while(c !=0)
  {
    P(SX) ;
    if(n==NB_PROCESS-1)
    {
       for(i=0 ; i<NB_PROCESS-1 ; i++)
         V(Barrier) ;
       n=0 ;
       V(SX) ;
    }
     else
     {
       n++ ;
       V(SX) ;
       P(Barrier) ;
     }
    c = traiter_image(flux) ;
  }
}
```