Programación de Sistemas
Examen Final 2do Término 2018

Nombre: _____ Paralelo: ____

Pregunta 1 (15 puntos)

El código abajo paraleliza la suma de un arreglo de *n* elementos cuyos valores son iguales a su índice y por lo tanto van del *0* a *n - 1*. Al final el hilo principal revisa si la suma total se calculó correctamente comparándola con la suma total esperada: $\frac{n}{2} \times (n-1)$

```
#define NTHREADS     4
#define ARRAYSIZE   100
#define SUMA_SERIE (ARRAYSIZE*(ARRAYSIZE-1))/2
#define ITERATIONS   ARRAYSIZE / NTHREADS

int  sum=0, a[ARRAYSIZE];
sem_t mutex;

void *do_work(void *tid)
{
  int i, start, *mytid, end;
  int mysum=0;

  mytid = (int *) tid;
  start = (*mytid * ITERATIONS);
  end = start + ITERATIONS;
  printf ("Hilo %d iterando desde %d hasta %d\n",
    *mytid,start,end-1);
  for (i=start; i < end ; i++) {
    a[i] = i;
    mysum = mysum + a[i];
  }

  sem_wait(&mutex);
  sum = sum + mysum;
  sem_post(&mutex);
  pthread_exit(NULL);
}
```

```
int main(int argc, char *argv[])
{
  int i, start, tids[NTHREADS];
  pthread_t threads[NTHREADS];

  sem_init(&mutex, 0, 1);    (+4 puntos)

  for (i=0; i<NTHREADS; i++) {
    tids[i] = i;
    pthread_create(&threads[i], NULL,
      do_work, (void *) &tids[i]);
  }

  for(i=0; i<NTHREADS; i++)   (+4 puntos)
    pthread_join(threads[i], NULL);

  printf ("Listo. Suma= %d\n", sum);

  if(sum == SUMA_SERIE)
    printf("Check suma OK!\n");
  else
    printf("BOOM!\n");

  pthread_exit (NULL);
}
```

Sin embargo, el programa nunca logra terminar correctamente porque el código contiene DOS errores que deben ser corregidos (ignore la omisión de *#include*).

1. Encuentre y corrija ambos errores. **(8 puntos)**
2. Muestre una posible salida del programa una vez corregidos los errores. **(7 puntos)**

Hilo 0 iterando desde 0 hasta 24
Hilo 3 iterando desde 75 hasta 99
Hilo 2 iterando desde 50 hasta 74
Hilo 1 iterando desde 25 hasta 49
Listo. Suma= 4950
Check suma OK!

Pregunta 2 (25 puntos)

Considere el siguiente código de dos programas llamados a.out y b.out. Asuma que ejecutamos el programa a.out de las siguiente forma:

./a.out 1 3

Muestre el diagrama de procesos, indicando que imprime cada proceso, el id de proceso y cuál ejecutable se está ejecutando (a.out o b.out). Asuma que los ID de proceso se asignan de manera secuencial, empezando desde 2000.

a.c

```
int main(int argc, char **argv){
  printf("2. PID: %d\n", getpid());
  pid_t b = getpid();
  int i = 0;

  pid_t h = fork();
  if(h > 0){
    if(strcmp("1", argv[1]) == 0){
      execl("b.out", "b.out", argv[2], NULL);
      printf("Fallo exec");
    }
    else{
      printf("7. PID: %d, PPID: %d\n",
                  getpid(), getppid());
      wait(NULL);
    }
  }
  else{
    printf("5. PID: %d, PPID: %d\n",
                getpid(), getppid());
    sleep(10000);
  }
}
```

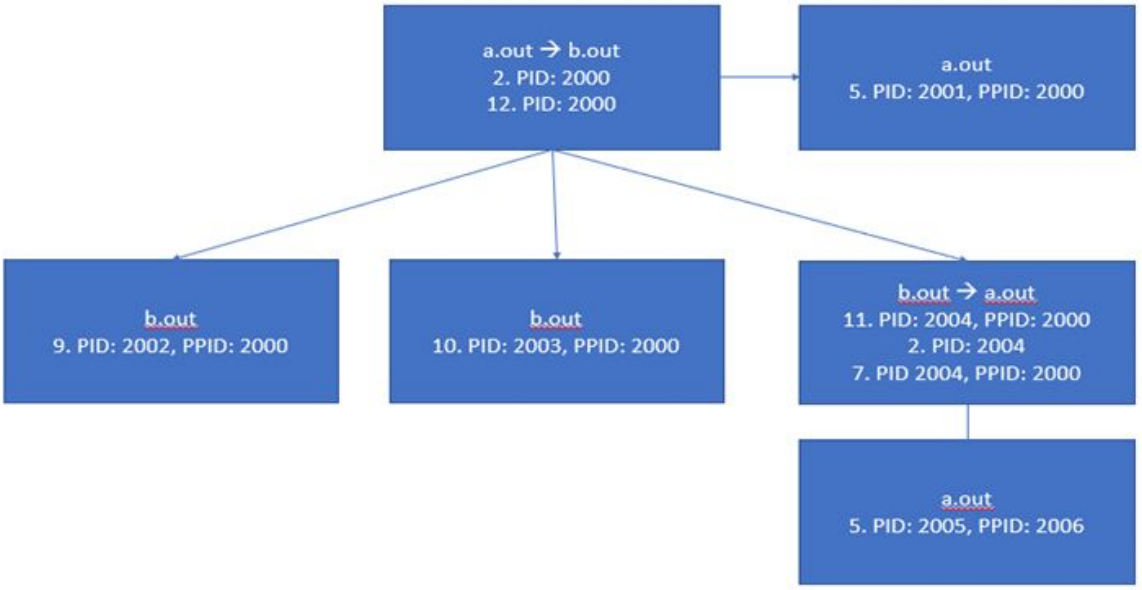
b.c

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

  int num = atoi(argv[1]);
  pid_t t = getpid();

  int i = 0;
  pid_t w = -1;
  int h = 8;
  for(i = 0; i < num; i++){
    if(t == getpid()){
      h++;
      w = fork();
    }
  }

  if(w == 0){

    printf("%d. PID: %d, PPID: %d\n", h,
                getpid(), getppid());
    if(h == 11){
      execl("a.out", "a.out", "0", "0", NULL);
      printf("Fallo exec\n");
    }
    sleep(10000);
  }
  else{
    printf("12. PID: %d\n", getpid());
    for(i = 0; i < num+1; i++){
      wait(NULL);
    }
  }
}
```

4 puntos por proceso
5 puntos por proceso 2004

Para los procesos:

- **1 punto por PID**
- **1 punto por PPID**
- **1 punto por mensaje**
- **1 punto por ejecutable**



Pregunta 3 (20 puntos)

El código abajo pertenece a un programa que imprime de manera constante el uso total de la memoria RAM en consola. Se ha usado el patrón de diseño productor - consumidor para implementar esta funcionalidad. El hilo productor “produce” un valor actualizado de uso de la memoria mientras que el hilo consumidor “consume” este valor y lo muestra en consola. Las llamadas a las funciones *sbuf_init, sbuf_remove* y *sbuf_insert* han sido removidas del código.

1. Inserte estas llamadas en el código en el sitio apropiado usando los parámetros apropiados.
2. ¿Con qué frecuencia el hilo consumidor mostraría el uso de la memoria en la consola? **50 ms (+5 puntos)**

```
typedef struct {
        float *buf;
        int n;
        int front;
        int rear;
        sem_t mutex;
        sem_t slots;
        sem_t items;
} sbuf_t;

void sbuf_init(sbuf_t *sp, int n);
float sbuf_remove(sbuf_t *sp);
void sbuf_insert(sbuf_t *sp, float item);

sbuf_t buff;

void *thread_consumidor(void *argp){

        float uso;
        setbuf(stdout, NULL);
        while(1){
            uso = sbuf_remove(&buff); (+5 puntos)
              printf("\rUso: %.1f",uso);
        }

        return NULL;
}

int main()
{
        int slots = 10;
        sbuf_init(&buff, slots); (+5 puntos)
        pthread_t con, prod;

        Pthread_create(&con, NULL, thread_consumidor, NULL);
        Pthread_create(&prod, NULL, thread_productor, NULL);
        Pthread_join(con, NULL);
        Pthread_join(prod, NULL);
}
```

```
void *thread_productor(void *argp){

        int memTotal; //Memoria total en el sistema
        int memFree; //Memoria libre en el sistema
        float memUso; //Porcentaje de memoria en uso
        FILE *meminfoFile;

        //Uso /proc/meminfo para obtener información de la memoria
        meminfoFile = fopen ("/proc/meminfo","r");
        if (meminfoFile == NULL)
                unix_error("No se pudo abrir /proc/meminfo");

        setbuf(meminfoFile, NULL);
        //Lee las primeras dos lineas de /proc/meminfo
        while( fscanf(meminfoFile, "MemTotal: %d kB MemFree: %d kB ",
          &memTotal, &memFree) == 2) {
                memUso = (((float) memTotal - (float) memFree)/
                (float) memTotal) * 100.0;
                sbuf_insert(&buff, memUso); (+5 puntos)
                //Regresa el puntero del archivo al inicio
                rewind(meminfoFile);
                usleep(50000);
        }

        fclose(meminfoFile);
        return NULL;
}
```

Pregunta 4 (25 puntos)

Un socket TCP/IP se almacena en la siguiente estructura en la interfaz de sockets:

```
struct sockaddr_in  {
    unsigned short  sin_family;
    unsigned short  sin_port;
    struct in_addr  sin_addr;
    unsigned char   sin_zero[8];
};
```

1. Muestre cómo se representaría en memoria byte por byte en hexadecimal el socket 127.0.0.1:3306 (asuma que se almacena en una variable tipo *struct sockaddr_in* y que el campo *sin_family* tiene el valor de *AF_INET = 2*).

todo se almacena en big endian

sin_family		***sin_port***		***sin_addr***				***sin_zero***							
00	02	0C	EA	7F	00	00	01	00	00	00	00	00	00	00	00

sin_family +3
sin_port +6
sin_addr +12
sin_zero +2

2. ¿Cuál es el objetivo del campo *sin_zero*?

Padding para mantener igual tamaño que sockaddr (+2)

Pregunta 5 (15 puntos)

Considere la siguiente llamada a open:

```
int fd = open("../archivo", O_WRONLY | O_TRUNC | O_CREAT, 0457);
```

Responda las siguientes preguntas:

1. Si el archivo existe, ¿qué va a suceder con el contenido del mismo?

El archivo se truncará (+3)

2. Si el archivo no existe ¿Qué pasará?

Se crea un archivo nuevo (+3)

3. Indique, para los usuarios mostrados abajo, que podrán hacer con el archivo (leer, escribir, ejecutar):
    a. Juan (user): Leer (+3)
    b. Carlos (grupo): Leer y ejecutar (+3)
    c. Maria (otros): Leer, escribir y ejecutar (+3)