Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencias de la Computación
R-322 — Sistemas Operativos I

Nombre y Apellido:

Exámen ~~Final~~

Ej. 1. Se quiere implementar un servidor multihilo para linux que atienda pedidos de varios clientes a través de la red, usando sockets y multiplexación de I/O con epoll. Vimos en la práctica que hay una serie de tareas que debemos hacer. Se pide completar el esqueleto de código que implementa un servidor multihilo. No es necesario escribir el código, puede utilizar las tareas que se listan a continuación; también puede utilizar funciones auxiliares que incluya alguna de las tareas[1] y también agregar tareas que no estén en la lista ó puede escribir directamente el código en C.

Crear un socket de escucha

agrear el socket a la instancia de epoll

(*) crear una instancia (local/global) de epoll

esperar un evento

ejecutar el handler de pedido del cliente

crear threads

```
#include <sys/epoll.h>
#include <sys/resource.h>
#include <pthread.h>

int main(int argc, char **argv)
{
    int nproc = sysconf(_SC_NPROCESSORS_ONLN) :
    pthread_t th[nproc];
        ...
        ...
    for (int i = 0; i < nproc; i++)
        pthread_join(th[i], NULL);
    return 0;
}
```

Ej. 2. Observe el siguiente programa:

```
1  int main(void)
2 {
3     int i;
4     int status;
```

[1](*) tachar lo que no corresponda entre (local/global).

```
5     int fd[2];
6     pipe(fd);
7     for (i = 0; i < 3; i++)
8     {
9         pid_t pid = fork();
10        if (pid == 0)
11        {
12            close(fd[0]);
13            char buff[5] = "HOLA\n";
14            int k = 0;
15            k = write(fd[1], &buff, 5);
16            if (k>0)
17                printf("Soy el hijo %d y salude a mi padre %d\n", getpid(), getppid());
18
19        }
20        else
21        {
22            close(fd[1]);
23            char buff2[5];
24            int r;
25            r = read(fd[0], buff2, 5);
26            if (r >0)
27                printf("Soy el padre %d, mi hijo %d me dijo %s", getpid(), pid, buff2);
28            wait(&status);
29        }
30    }
31    return 0;
32}
```

El resultado de la ejecución fue el siguiente:

```
Soy el hijo 9261 y salude a mi padre 9260
Soy el padre 9260, mi hijo 9261 me dijo HOLA
Soy el hijo 9262 y salude a mi padre 9261
Soy el hijo 9263 y salude a mi padre 9262
Soy el padre 9260, mi hijo 9265 me dijo HOLA
Soy el padre 9260, mi hijo 9267 me dijo HOLA
```

a) Explique el comportamiento del programa. ¿Cuantos procesos fueron creados? ¿Por qué?

b) ¿Todos los hijos pudieron saludar a su padre y todos los padres recibieron el saludos de sus hijos? Explique por qué sucede esto.

c) ¿Cómo se modificaría la salida si en la línea 18 agregamos un exit(0) ?

Ej. 3. Supongamos que la función `tests_and_set` es una operación atómica.

```
int value = 0;
int hello = 0;
int test_and_set(int* value) {
  int old_value = *value;
  *value = 1;
  return old_value;
```

```
}

void print_hello() {
  while (test_and_set(&value));
  hello += 1;
  printf("Child thread: %d\n", hello);
  value = 0;
  pthread_exit(0);
}

void main() {
  pthread_t thread1, thread2;
  pthread_create(&thread1, NULL, &print_hello, NULL);
  pthread_create(&thread2, NULL, &print_hello, NULL);
  while (test_and_set(&value));
  printf("Parent thread: %d\n", hello);
  value = 0;
  return;
}
```

Asumiendo la siguiente secuencia de eventos:

1. El proceso `main` comienza a correr y crea los dos threads, después ocurre un cambio de contexto.
2. El scheduler pone al `thread2` a correr hasta que incrementa la variable `hello`. Después vuelve a haber un cambio de contexto.
3. El `thread1` corre hasta que es cambiado por el scheduler.
4. El `thread main` resume su ejecución y ejecuta hasta que es cambiado por el scheduler.
5. El `thread2` corre hasta completar su ejecución y termina (hace `pthread_exit`).
6. El `thread main` corre hasta completar su ejecución pero no termina (no hace `return`).
7. El `thread1` corre hasta terminar.

Responda:

a) Indicar la evolución de cada thread, es decir, comentar cuando los threads se "bloquean" y se "desbloquean" y los valores que toman las variables `value` y `hello` en cada momento.

b) Para los pasos donde se invoca a `test_and_set`, ¿que va a devolver?

c) ¿Cúal es la salida del programa?

d) ¿Cúal es el mayor inconveniente de esta implementación de sincronización?

Ej. 4. ¿Cúales de los siguientes componentes del estado de un programa se comparten entre los threads de un proceso?

Valores de registros, Heap, Variables globales, Stack

Ej. 5. ¿Qué son y cómo se relacionan entre sí los conceptos de región crítica, *race condition* y exclusión mutua?