CC3301 Programación de Software de Sistemas

Examen – Semestre Otoño 2013 – Prof.: Luis Mateu

Pregunta 1

En una calle hay 5 estacionamientos al lado derecho y ninguno al lado izquierdo. Se identifican como 0, 1, 2, 3 y 4. Un auto necesita solo 1 estacionamiento, una camioneta necesita 2 que estén contiguos, un camión 3 contiguos pero si tiene remolque necesita los 5 estacionamientos.

Los automovilistas son representados por threads que invocan las funciones *reservar* y *liberar* para controlar el uso de los estacionamientos. Cuando un automovilista llega invoca la función *reservar* indicando su nombre (*nom*) y cuantos estacionamientos contiguos necesita (*k*). Si hay *k* estacionamientos contiguos disponibles, esta función los reserva y retorna de inmediato la identificación del primer estacionamiento en la serie otorgada. De lo contrario, *reservar* espera hasta que hayan *k* estacionamientos contiguos disponibles. Cuando un automovilista se va invoca la función *liberar* indicando su nombre (*nom*). Esta función libera todos los estacionamientos de la serie otorgada previamente a ese automovilista con *reservar*, y por lo tanto pueden ser otorgados a otros vehículos.

Programe en C las funciones *reservar* y *liberar*. Sus encabezados son:

```
int reservar(char *nom, int k);
void liberar(char *nom);
```

Importante: En *reservar* Ud. debe *crear una copia del nombre*, porque el nombre original será destruido en el llamador. Del mismo modo, en la función *liberar*, debe liberar el espacio de la copia.

Vea el ejemplo de uso de la pregunta 2. Le servirá para entender mejor qué deben hacer estas funciones. No es necesario evitar la hambruna. No necesita validar que un automovilista haya llamado *reservar* antes de *liberar*. Le será útil programar primero una función *ubicar*, con la misma funcionalidad que *reservar*, excepto que retorna -1 cuando no hay *k* estacionamientos contiguos en vez de esperar. Priorice la simplicidad por sobre la eficiencia.

Pregunta 2

Programe un servidor y un cliente para la reserva de los estacionamientos de la pregunta 1 por Internet. El servidor se llama `calle`, corre siempre en *localhost* y escucha a los clientes en el puerto 3000. El cliente se llama `auto`, el que invocan los choferes para reservar o liberar los estacionamientos. La tabla de arriba a la derecha muestra un ejemplo de uso. Juan, Eva, Pato y Ana son choferes. Las filas están ordenadas cronológicamente.

En la celda A, Juan reserva el estacionamiento 0. En B, Eva reserva 1 y 2. En C, Pato solicita 3 estacionamientos que no hay, por lo tanto espera. En D, Ana reserva el estacionamiento 3. En E, Eva libera 1 y 2. Ahora hay 3 estacionamientos libres pero no son contiguos por lo que Pato sigue esperando. En F, Juan libera 0. Ahora sí hay 3 estacionamientos contiguos, 0, 1 y 2, que se otorgan a Pato y por lo tanto su espera termina (ver celda G).

Hint: Use en el servidor las funciones de la pregunta 1. Suponga que no se producen errores de *sockets* o entrada/salida (*j_bind*, *j_connect*, *read*, *write*, etc.).

Juan	Eva	Pato	Ana

`% auto juan r 1`
`0`
`%` *(A)*			

	`% auto eva r 2`
	`1`
	`%` *(B)*		

		`% auto pato r 3` *(espera, C)*	

			`% auto ana r 1`
			`3`
			`%` *(D)*

	`% auto eva l` `%` *(E)*		
`% auto juan l` `%` *(F)*		`0` `%` *(G)*	

Pregunta 3

i. Complete en este programa los espacios marcados como … de modo que el programa no arroje errores de compilación o *warnings*.

```
typedef struct { … } Str;
typedef … Fun …;
int proc(Fun f, Str *p, char *q) {
  … r= &q;
  … s= &r;
  p->d= **s;
  return (*f)(p);
}
```

ii. Un proceso se comunica con 2 *sockets* que entregó *j_accept*. Sus descriptores son *fd1* y *fd2*. Se sabe que llegarán datos, pero no se sabe si lo harán por *fd1* o *fd2*. Escriba un trozo de código que lea los datos en cuanto lleguen. No puede usar *threads*. Observe que si lee de *fd1*, pero los datos llegan por *fd2*, el proceso se quedará bloqueado indefinidamente, y viceversa.

iii. Considere la función *leerInt* de más abajo. En ella si *p* apunta a una dirección inválida, se gatillará la señal SIGSEGV, es decir *segmentation fault*, lo que normalmente termina el programa.

```
int leerInt(int *p) {
  return *p;
}
```

Modifique la función de tal forma que si la dirección es inválida, *leerInt* entregue 0 en vez de terminar el programa. *Hint*: Ud. debe capturar la señal SIGSEGV y usar *setjmp*/*longjmp*.