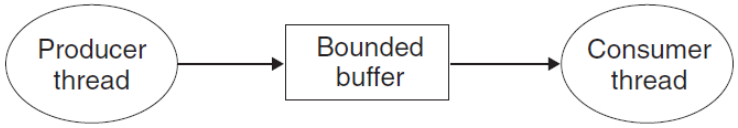
Examen Final de Programación de Sistemas

1er término 2017 – 2018

Nombre: _____ **Paralelo**: _____

Pregunta 1 (25 puntos)

En el problema **Productor – Consumidor** existen dos hilos: un hilo productor, el cual inserta ítems en un *buffer* compartido mientras los va produciendo, y un hilo consumidor el cual remueve los ítems del *buffer* para consumirlos.



Para implementar e inicializar el *buffer* tenemos el siguiente código:

```
typedef struct {
    int *buf;          /* Buffer array */
    int n;             /* Maximum number of slots */
    int front;         /* buf[(front+1)%n] is first item */
    int rear;          /* buf[rear%n] is last item */
    sem_t mutex;       /* Protects accesses to buf */
    sem_t slots;       /* Counts available slots */
    sem_t items;       /* Counts available items */
} sbuf_t;
```

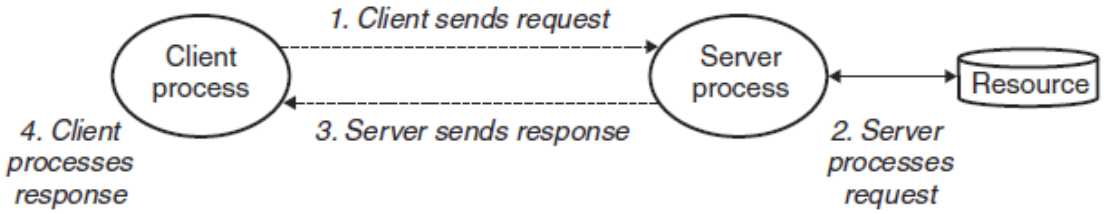
```
/* Create an empty, bounded, shared FIFO buffer with n slots */
void sbuf_init(sbuf_t *sp, int n)
{
    sp->buf = Calloc(n, sizeof(int));
    sp->n = n;                       /* Buffer holds max of n items */
    sp->front = sp->rear = 0;        /* Empty buffer iff front == rear */
    Sem_init(&sp->mutex, 0, 1);      /* Binary semaphore for locking */
    Sem_init(&sp->slots, 0, n);      /* Initially, buf has n empty slots */
    Sem_init(&sp->items, 0, 0);      /* Initially, buf has zero data items */
}
```

Escriba el código de la función para insertar ítems de manera segura y sincronizada al *buffer* usando el siguiente prototipo para la función:

```
/* Insert item onto the rear of shared buffer sp */
void sbuf_insert(sbuf_t *sp, int item)
{
    P(&sp->slots);                          /* Wait for available slot */
    P(&sp->mutex);                          /* Lock the buffer */
    sp->buf[(++sp->rear)%(sp->n)] = item;   /* Insert the item */
    V(&sp->mutex);                          /* Unlock the buffer */
    V(&sp->items);                          /* Announce available item */
}
```

Pregunta 2 (15 puntos)

Complete el diagrama del modelo de programación cliente-servidor y describa cada una de sus etapas.



1. When a client needs service, it initiates a transaction by sending a *request* to the server. For example, when a Web browser needs a file, it sends a request to a Web server.
2. The server receives the request, interprets it, and manipulates its resources in the appropriate way. For example, when a Web server receives a request from a browser, it reads a disk file.
3. The server sends a *response* to the client, and then waits for the next request. For example, a Web server sends the file back to a client.
4. The client receives the response and manipulates it. For example, after a Web browser receives a page from the server, it displays it on the screen.

Pregunta 3 (5 puntos)

El siguiente gráfico ilustra un ejemplo de uso de sockets para comunicación en un modelo cliente-servidor. ¿Quién maneja el puerto bien-conocido y quien el puerto efímero?

a) Cliente: puerto efímero asignado manualmente por el usuario; Servidor: Puerto bien-conocido asignado al servicio.

b) Cliente: puerto efímero asignado por el *kernel*; Servidor: Puerto bien-conocido asignado al servicio.

c) Cliente: Puerto bien-conocido asignado al servicio; Servidor: puerto efímero asignado manualmente por el usuario.

d) Cliente: Puerto bien-conocido asignado al servicio; Servidor: puerto efímero asignado por el *kernel*.

Pregunta 4 (25 puntos)

¿Cuál es la salida del siguiente código?

```
1   pid_t pid;
2   int counter = 2;
3
4   void handler1(int sig) {
5       counter = counter - 1;
6       printf("%d", counter);
7       fflush(stdout);
8       exit(0);
9   }
10
11  int main() {
12      signal(SIGUSR1, handler1);
13
14      printf("%d", counter);
15      fflush(stdout);
16
17      if ((pid = fork()) == 0) {
18          while(1) {};
19      }
20      kill(pid, SIGUSR1);
21      waitpid(-1, NULL, 0);
22      counter = counter + 1;
23      printf("%d", counter);
24      exit(0);
25  }
```

//Este *handler* es ejecutado solo por el hijo

//Registra el *handler* de la señal SIGUSR1

//Crea proceso hijo

//Dentro del proceso hijo

//Padre envía la señal SIGUSR1 al proceso hijo

213

Pregunta 5 (20 puntos)

Determine la salida del siguiente programa. ¿Existe una condición de carrera? ¿Por qué?

```
int value=1;

void *Hilo1(void *argus)
{
  int cont=0;
  while (cont < 3){
        value = value + 2*value;
        cont++;
        sleep(3);
  }
  pthread_exit(NULL);
}

void *Hilo2(void *argus)
{
  int cont=0;

  while (cont < 3){
        value = value + 2*value;
        cont++;
        sleep(1);
  }

  pthread_exit(NULL);
}

int main()
{
  pthread_t myThread1, myThread2;
  int aux=0;

  pthread_create(&myThread1, NULL, &Hilo1, (void *)&aux);
  pthread_create(&myThread2, NULL, &Hilo2, (void *)&aux);

  pthread_join(myThread1, NULL);
  pthread_join(myThread2, NULL);

  printf("\nRespuesta: %d\n", value);

  return 0;
}
```

Respuesta: 729

La variable global *value* es compartida y accedida simultáneamente por dos hilos y en la ausencia de un semáforo **sí** se podría dar una condición de carrera.

Pregunta 6 (5 puntos)

¿Una transacción cliente servidor es equivalente a una transacción de base de datos?

a) Sí, sus transacciones comparten las mismas propiedades
b) Sí, una transacción cliente servidor es siempre atómica
c) No, sus transacciones no comparten las mismas propiedades
d) Sí, considerando que una transacción implica procesos complejos por parte del servidor

Pregunta 7 (5 puntos)

Seleccione la afirmación correcta. ¿Por qué el protocolo TCP es inadecuado para el transporte de tráfico multimedia en comparación con UDP?

a) Porque TCP se encarga del transporte de paquetes sin garantizar la entrega de los mismos.
b) TCP se hace cargo de pérdidas mediante la retransmisión de paquetes perdidos, pero no garantiza un tiempo de llegada.
c) Porque el protocolo TCP utiliza el protocolo IP por lo cual es inadecuado para tráfico multimedia.
d) UDP se hace cargo de pérdidas mediante la retransmisión de paquetes perdidos, pero no garantiza un tiempo fijo de llegada.