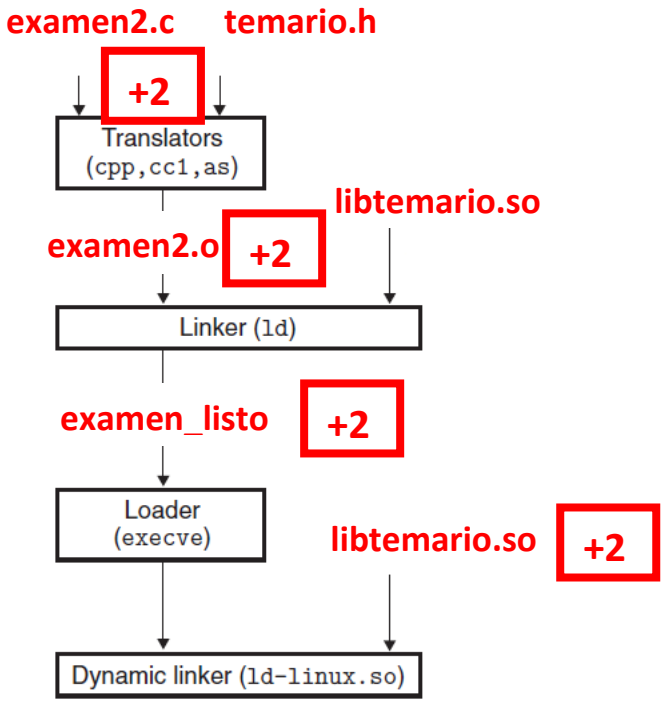
Examen Final Programación de Sistemas

2do Término 2017 – 2018

Nombre: _____

Pregunta 1 (10 puntos)

Dado un programa escrito en lenguaje C, llamado "`examen2.c`" y que requiere una librería llamada "`libtemario.so`" con su respectiva cabecera "`temario.h`". Complete el siguiente diagrama para obtener el ejecutable "`examen_listo`", indicando las entradas y salidas de cada una de las fases mostradas. Explique, ¿por qué es requerida la fase del "Dynamic linker" y cuál es el resultado de la misma?



La fase del dynamic linker es requerida para acceder y usar el código ejecutable en libtermario.so en tiempo de ejecución. La librería libtemario.so es dinámica y por lo tanto su contenido no está incluido en el ejecutable *examen_listo*. + 2

Pregunta 2 (20 puntos)

Considere el siguiente código. Muestre el diagrama de procesos (que proceso es padre de quien, y que imprime dicho proceso).

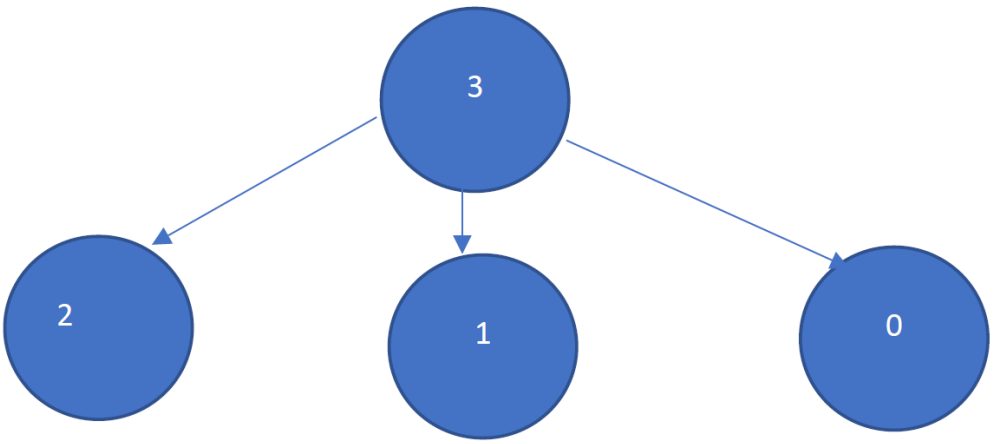
```
int main(void)
{

	pid_t pid;

	printf("3\n");
	pid_t a = getpid();

	int i = 0;
	for(i = 0; i < 3; i++){
		int f = getpid();
		if(f == a){
			pid = fork();
			if(pid == 0)
				printf("%d\n", i);
		}
	}
	if(pid >  0){
		int stat;
		for(i = 3; i > 0; i--)
			wait(&stat);
	}
	else
		while(1);
}
```

Los hijos pueden estar en cualquier orden.



1. **(+10) Muestra diagrama correcto**
2. **(+10) Muestra salidas correctas**

Pregunta 3 (20 puntos)

Se ha implementado una simulación del juego del "¡Chantón, chantón!". En dicho juego, todos los participantes tratan de completar un conjunto de palabras a partir de una letra, por ejemplo: Si se elige la letra A, cada jugador debe escribir un nombre, un apellido, un animal, una cosa, etc., todas empezando con la letra A.

Cuando un jugador completa todas las palabras requeridas, dice una palabra escogida de parada y el resto de jugadores deben detener su juego. Y se asigna puntos por cada una de las palabras escogidas correctamente.

En la simulación, se utilizarán 4 jugadores, en 5 rondas. En cada ronda cada jugador debe completar 5 palabras o columnas. Cada jugador es representado por un hilo de ejecución (thread) y cuando se inicia, esperará un tiempo aleatorio por cada una de las columnas (palabras) especificadas. Cuando un hilo (jugador) completa las 3 columnas, los demás hilos deben detenerse.

Se ha utilizado la siguiente implementación: (Ver reverso)

Pero al momento de ejecutar se presenta la siguiente salida:

```
----- Ronda # 0
Puntos Jugador # 0: 0
Puntos Jugador # 1: 0
Puntos Jugador # 2: 0
Puntos Jugador # 3: 0
1 dice: CHANTON CHANTON!
0 dice: CHANTON CHANTON!
2 dice: CHANTON CHANTON!
3 dice: CHANTON CHANTON!
```

Indique qué líneas modificaría o agregaría para corregir el programa y que se ejecute correctamente. Asuma que no tiene problemas de sincronización.

1. **(+ 10) Busca una forma en la cual el hilo principal espera al resto.**
    a. **(+2 extra) Elimina línea 26: pthread_exit(NULL)**
2. **(+ 10) Busca una forma en la cual el hilo ganador detiene al resto.**

Continuación Pregunta 3

```c
#include "csapp.h"
#define JUGADORES 4
#define RONDAS 5
#define COLUMNAS 3

void *thread(void *vargp);

int puntos[JUGADORES];
pthread_t tid[JUGADORES];

int main()
{
        srand (time(NULL));
        int ronda, i, *ptr;

        for (ronda = 0; ronda < RONDAS; ronda++){
                printf("----- Ronda # %d\n", ronda);
                for (i = 0; i < JUGADORES; i++) {
                        ptr = Malloc(sizeof(int));
                        *ptr = i;
                        Pthread_create(&tid[i], NULL, thread, ptr);
                }

                for (i = 0; i < JUGADORES; i++)
                        Pthread_join(tid[i], NULL);

                for (i = 0; i < JUGADORES; i++)
                        printf("Puntos Jugador # %d: %d\n", i, puntos[i]);
                Pthread_exit(NULL);
        }
        exit(0);
}

/* Rutina de jugadores */
void *thread(void *vargp)
{
        int piensa, columna, myid = *((int *)vargp);
        Free(vargp);
        for (columna = 0; columna < COLUMNAS; columna++){
                piensa = rand() % 5 + 1;  // Número aleatorio entre 1 y 5
                sleep(piensa);
                puntos[myid]+=1;
        }
        printf("%d dice: CHANTON CHANTON!\n", myid);
        for (int i = 0; i < JUGADORES; i++)
           if (i != myid)
              Pthread_cancel(tid[i]);

        return NULL;
}
```

1

2

Pregunta 4 (10 puntos)

Considere que tiene un sistema que atiende a varios clientes. Para esto, Ud. tiene varios hilos de atención, y un hilo que recibe a los clientes, y los pone en una cola. El hilo de atención siempre toma un cliente de la cola a la vez, lo atiende, y luego busca otro cliente. A continuación se muestra una primera implementación. Determine si hay condiciones de carrera, y si las hay, solucionelas usando semáforos.

```
#define MAX_CLIENTES 50
int clientes = 0;
int cola[MAX_CLIENTES] = {0};


void atender(){
	while(1){
		if(clientes > 0){

			Cliente cliente;
			decolar(cola, &cliente);
			clientes--;
			//Procesar el cliente, toma un tiempo aleatorio
			procesar(cliente);

		}
	}
}

void recibir_cliente(){
	while(1){
		if(clientes < 50){
			//Toma tiempo aleatorio
			Cliente cliente = nuevo_cliente();
			encolar(cola, &cliente);
			clientes++;
		}
	}
}
```

1. **(+3) Inicializa el/los semáforos correctamente**
2. **(+2) Protege el acceso a la cola**
3. **(+2) Protege el acceso a la variable clientes**
4. **(+3) Protege el acceso en ambas funciones**
5. **(-8) Coloca semáforos fuera del while**
6. **(-1) Usa más de un semáforo**

1

```
sem_t s;
sem_init(&s,0,1);
void atender(){
	while(1){
		if(clientes > 0){
			Cliente cliente;
			P(&s);
			decolar(cola, &cliente);
			clientes--;
			V(&s);
			//Procesar el cliente, toma un tiempo aleatorio
			procesar(cliente);
		}

	}
}

void recibir_cliente(){
	while(1){
		P(&s);
		if(clientes < 50){
			//Toma tiempo aleatorio
			Cliente cliente = nuevo_cliente();


			encolar(cola, &cliente);
			clientes++;
		}
		V(&s);
	}
}
```

2 - 4

Pregunta 5 (15 puntos)

En una aplicación cliente - servidor se tiene la siguiente función en el servidor la cual se encarga de enviar un archivo a un cliente ya conectado (ver reverso).

Crear la respectiva función en el cliente que reciba el archivo y lo guarde en un archivo con el nombre “archivo_recibido.dat”. La función tiene el siguiente prototipo:

```
//Retorna el tamaño en bytes del archivo recibido, 0 si hubo error
int recibir_archivo_servidor(int connfd);
```

Referencia

Las funciones RIO que se pueden usar con las siguientes:

```
//Wrapper read()
ssize_t Rio_readn(int fd, void *usrbuf, size_t n);
//Wrapper write()
void Rio_writen(int fd, void *usrbuf, size_t n);
//Inicializa un buffer con un descriptor de archivo
void Rio_readinitb(rio_t *rp, int fd);
/* Lee usando read() un buffer rp hasta encontrar un salto de línea '\n'
      Retorna el número de bytes leidos */
ssize_t Rio_readlineb(rio_t *rp, void *usrbuf, size_t maxlen);
```

```
#define MAXLEN 80
int recibir_archivo_servidor(int connfd)
{
        rio_t rp;
        ssize_t n,filesize = 0;
        FILE * fp;
        char line_buffer[MAXLEN];
        char *file_buffer;

        Rio_readinitb(&rp, connfd);

        //Lee tamaño archivo enviado desde enviar_archivo_cliente
        Rio_readlineb(&rp, line_buffer, MAXLEN);
        //Convierte la cadena de caracteres recibida a ssize_t
        filesize = (ssize_t) atoi(line_buffer);
        //Si es diferente de 0, se asume no hubo error
        if(filesize){
                //Crea archivo con acceso escritura
                fp = fopen("archivo_recibido.dat","w");
                //Crea un buffer temporal
                file_buffer = malloc(filesize);
                //Lee el archivo enviado desde enviar_archivo_cliente en el buffer
                n = Rio_readn(connfd, file_buffer, filesize);
                if(n > 0){
                        //Escribe el archivo
                        fwrite(file_buffer, 1, n, fp);
                }else
                        n = 0;

                free(file_buffer);
                fclose(fp);
        }else
                return 0;

        return (int) n;
}
```

1

2

3

4

1. **(+2) Declara e inicializa correctamente variables y buffers**
2. **(+5) Lee correctamente el tamaño del archivo, o valida el error, del servidor**
3. **(+5) Crea correctamente un archivo nuevo y copia datos del servidor**
4. **(+3) Libera correctamente los recursos creados**

Continuación Pregunta 5

```
//Retorna el tamaño en bytes del archivo enviado, 0 si hubo error
int enviar_archivo_cliente(const char *nombre_archivo, int connfd)
{
	int n = 0;
	struct stat sbuf;
	char buf[MAXLINE];
	char *filebuf;
	FILE * fp;

	if(stat(filename,&sbuf) < 0)
	{
		Rio_writen(connfd, "NO ENCONTRADO\n", 14);
	}else {
		n = sbuf.st_size;
		if(n > 0) {
			sprintf(buf, "%lu\n", n);
			Rio_writen(connfd, buf, n);
			fp = Fopen(nombre_archivo,"r");
			filebuf = Malloc(n);
			n = Fread(filebuf, 1, n, fp);
			if (n > 0)
				Rio_writen(connfd, filebuf, n);
			Free(filebuf);
			Fclose(fp);
		}else{
			Rio_writen(connfd, "ERROR\n", 6);
		}
	}

	return n;
}
```

Pregunta 6 (10 puntos)

En la interfaz de hilos Pthread, un hilo (thread) puede estar en dos estados distintos: *joinable* o *detached*. Explique en qué consisten estos estados y cuáles son las respectivas ventajas de cada uno proporcionando dos ejemplos.

1. **(+2) Un hilo en estado joinable puede ser esperado y su valor de retorno recogido por otro hilo usando pthread_join. Los recursos usados por el hilo son liberados una vez que el hilo ha sido recogido por otro hilo.**
2. **(+2) Un hilo en estado detached no puede ser esperado ni recogido por otros hilos, sus recursos son liberados por el kernel cuando el hilo termina.**
3. **(+2) La ventaja de un hilo en estado joinable es que es posible sincronizarse al mismo y obtener resultados de su trabajo.**
4. **(+1) Ejemplo: queremos calcular la suma total de un arreglo bastante grande. Dividimos el arreglo en porciones pequeñas para cada hilo, esperamos a los hilos y obtenemos su valor de retorno para finalmente sumarlos y obtener la suma total.**
5. **(+2) La ventaja de un hilo en estado detached es que ya no es responsabilidad de la aplicación esperar al hilo para liberal sus recursos.**
6. **(+1) Ejemplo: En una aplicación cliente – servidor queremos que cada cliente conectado sea atendido por un hilo diferente en estado detached. Al hilo principal que despacha las conexiones al resto de hilos no le interesa saber cuándo cada hilo ha terminado ni el resultado de su trabajo.**

Pregunta 7 (15 puntos)

En la interfaz de sockets, en el lado del servidor, se usan dos sockets, uno de escucha y otro de conexión. En el lado del cliente, tan solo se usa un socket, el de conexión. ¿Por qué esta diferencia? ¿Cuál es la ventaja de usar dos sockets en el lado del servidor?

1. **(+4)** En el servidor se usa un descriptor de archivo socket de escucha a nuevas conexiones, **se crea una vez y se mantiene durante el periodo completo de ejecución del servidor.**
2. **(+4)** En el servidor se usa un descriptor de archivo socket de conexión por cada cliente conectado al servidor, **se crea al establecerse una conexión con un cliente y se destruye cuando el cliente se desconecta.**
3. **(+5) La ventaja de este esquema es que permite la concurrencia de conexiones en el servidor.**
4. **(+2) Se tiene un descriptor de archivo de socket diferente por cliente y por lo tanto cada descriptor puede ser asignado a un hilo o proceso diferente para ser despachado de manera concurrente.**