# Programación de Sistemas
# Examen Final 1er Término 2018

Nombre: _____ Paralelo: ____

## Pregunta 1 (25 puntos)

Considere el siguiente código. Muestre el diagrama de procesos, y que imprime cada proceso. Asuma que los PIDs se asignan de manera secuencial, empezando desde 1000. El programa (a.out) se ejecuta con el siguiente comando:

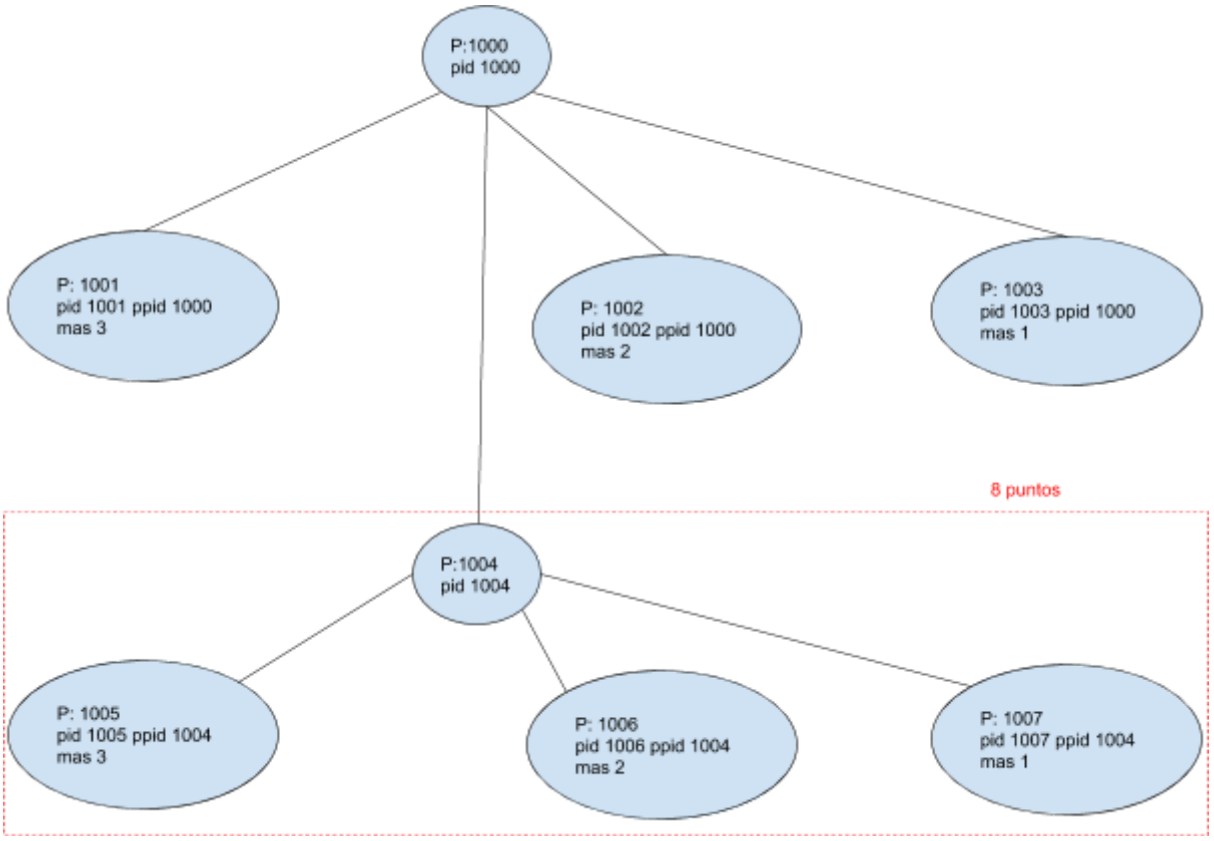
**./a.out 1**

```
#define MAX 4
pid_t mas(int i, int execute){
	pid_t t = fork();
	if(t == 0){
		if(i != 0){
			printf("pid %d ppid %d\n",getpid(),getppid());
			printf("mas %d\n", i);
			while(1);
		}
		else{ //8 puntos por el execl
			if(execute){
				//execl(comando,argv[0],argv[1],...)
				execl("./a.out","./a.out", "0", NULL);
			}
		}
	}else
		return t;
	return -1;
}

int main(int argc, char** argv){
	pid_t d = getpid();
	printf("pid %d\n",d);
	pid_t back[4] = {-1};

	for(int j = MAX-1; j >= 0; j--){
		if(d == getpid()){
			back[j] = mas(j, atoi(argv[1]));
		}
	}
	for(int i = 0; i < MAX; i++)
		waitpid(back[i], NULL,0);
	return 0;
}
```

Respuesta:




Correcto número de procesos: 5 puntos
Salida correcta en cada proceso: 12 puntos
Correcta interpretación de execl: 8 puntos

## Pregunta 2 (20 puntos)

```
#define N 3
void *thread(void *vargp);

char **ptr;

int main()
{
	int i;
	pthread_t tid;
	char *msgs[N] = {
	"Hola desde foo",
	"Hola desde bar",
	"Hola desde allá",
	};

	printf("Hola desde aquí\n");
```

```
        ptr = msgs;
        for (i = 0; i < N; i++)
              pthread_create(&tid, NULL, thread, (void *)i);

        pthread_join(tid,NULL);
        printf("Hola desde main\n");
        pthread_exit(NULL);
        printf("Final\n");
}

void *thread(void *vargp)
{
        int myid = (int) vargp;
        static int cnt = 0;
        printf("[%d]: %s (cnt=%d)\n", myid, ptr[myid], ++cnt);
        return NULL;
}
```

El siguiente código tiene varias posibles salidas al ser ejecutado las cuales dependen del orden en el cual el kernel decida agendar los hilos creados. Escriba 4 posibles salidas.

Respuesta:

| Ejemplo 1 | Ejemplo 2 | Ejemplo 3 | Ejemplo 4 |
|---|---|---|---|
| Hola desde aquí<br>[0]: Hola desde foo (cnt=1)<br>[2]: Hola desde allá (cnt=2)<br>[1]: Hola desde bar (cnt=3)<br>Hola desde main | Hola desde aquí<br>[0]: Hola desde foo (cnt=1)<br>[1]: Hola desde bar (cnt=2)<br>[2]: Hola desde allá (cnt=3)<br>Hola desde main | Hola desde aquí<br>[1]: Hola desde bar (cnt=1)<br>[2]: Hola desde allá (cnt=2)<br>Hola desde main<br>[0]: Hola desde foo (cnt=3) | Hola desde aquí<br>[2]: Hola desde allá (cnt=1)<br>Hola desde main<br>[0]: Hola desde foo (cnt=2)<br>[1]: Hola desde bar (cnt=3) |

- ***Hola desde aquí*** va al principio (+2)
- ***Final*** está ausente (+5)
- ***[2]: Hola desde allá*** va siempre antes de ***Hola desde main*** (+5)
- Correcta posición de ***[0], [1]*** y ***[2]*** (+5)
- Variar el valor de ***cnt*** mostrando que es una variable compartida (+3)

## Pregunta 3 (15 puntos)

```
1   int open_listenfd(int port)
2   {
3       int listenfd, optval=1;
4       struct sockaddr_in serveraddr;
5
6       /* Create a socket descriptor */
7       if ((listenfd = 1:                                    ) < 0)
8           return -1;
9
10      /* Eliminates "Address already in use" error from bind */
11      if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
12                     (const void *)&optval , sizeof(int)) < 0)
13          return -1;
14
15      /* Listenfd will be an end point for all requests to port
16         on any IP address for this host */
17      bzero((char *) &serveraddr, sizeof(serveraddr));
18      serveraddr.sin_family = AF_INET;
19      serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
20      serveraddr.sin_port = htons((unsigned short)port);
21      if ( 2:                                                ) < 0)
22          return -1;
23
24      /* Make it a listening socket ready to accept connection requests */
25      if ( 3:                          < 0)
26          return -1;
27      return listenfd;
28  }
```

La función *open_listenfd* que se presenta arriba es una función de ayuda en el proceso servidor. El parámetro de entrada es el puerto TCP en donde el servidor va a escuchar conexiones nuevas con procesos clientes. La función retorna un descriptor de socket listo para ser usado con la función *accept*. Complete las tres llamadas a funciones de la interfaz de sockets, con sus respectivos parámetros, que se muestran suprimidas en el código arriba presentado.

Respuesta:

```c
int open_listenfd(int port)
{
    int listenfd, optval=1;
    struct sockaddr_in serveraddr;

    /* Create a socket descriptor */
    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        return -1;

    /* Eliminates "Address already in use" error from bind */
    if (setsockopt(listenfd, SOL_SOCKET, SO_REUSEADDR,
                   (const void *)&optval , sizeof(int)) < 0)
        return -1;

    /* Listenfd will be an end point for all requests to port
       on any IP address for this host */
    bzero((char *) &serveraddr, sizeof(serveraddr));
    serveraddr.sin_family = AF_INET;
    serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
    serveraddr.sin_port = htons((unsigned short)port);
    if (bind(listenfd, (SA *)&serveraddr, sizeof(serveraddr)) < 0)
        return -1;

    /* Make it a listening socket ready to accept connection requests */
    if (listen(listenfd, LISTENQ) < 0)
        return -1;
    return listenfd;
}
```

# Pregunta 4 (20 puntos)

Ud. está desarrollando un nuevo software, y ha decidido modularizar su código en varias librerías. Su software será usado en dispositivos con recursos de hardware limitados (CPU, memoria, conectividad). Al compilar su ejecutable y enlazarlo estáticamente, Ud. se da cuenta que el tamaño de su ejecutable es muy grande. Responda las siguientes preguntas:

1. ¿Por qué el ejecutable resultó muy grande?

Todas las funciones en las librerías estáticas que son usadas en el software son copiadas directamente en el ejecutable. (+7)

2. ¿Qué podría hacer para reducir el tamaño del ejecutable?

Usar librerías dinámicas. (+7)

3. ¿Tiene alguna otra ventaja su solución? Explique.

- No es necesario volver a recompilar y redistribuir el programa entero si una librería dinámica es corregida. (+6)
- Librerías y programa son distribuidas independientemente. (+6)

- Uso eficiente de la memoria y disco si otros programas usan las mismas librerías. (+6)

## Pregunta 5 (20 puntos)

```
/* Global variables */
int readcnt;    /* Initially = 0 */
sem_t mutex, w; /* Both initially = 1 */

void reader(void)
{
    while (1) {
        Implemente el código del reader...
    }
}
```

```
void writer(void)
{
    while (1) {
        P(&w);

        /* Critical section */
        /* Writing happens  */

        V(&w);
    }
}
```

El código de arriba muestra la **primera versión** del problema lectores - escritores: se favorece al lector y se requiere que ningún lector espere a menos que un escritor esté usando ya el recurso. Los lectores pueden acceder al recurso (*Critical section*) de manera concurrente siempre y cuando un escritor no esté escribiendo en la sección crítica. Los escritores necesitan acceso exclusivo a la sección crítica. Implemente el código faltante en la función reader.

Respuesta:

```
/* Global variables */
int readcnt;    /* Initially = 0 */
sem_t mutex, w; /* Both initially = 1 */

void reader(void)
{
    while (1) {
        P(&mutex);
        readcnt++;
        if (readcnt == 1) /* First in */
            P(&w);
        V(&mutex);

        /* Critical section */
        /* Reading happens  */

        P(&mutex);
        readcnt--;
        if (readcnt == 0) /* Last out */
            V(&w);
        V(&mutex);
    }
}
```

```
void writer(void)
{
    while (1) {
        P(&w);

        /* Critical section */
        /* Writing happens  */

        V(&w);
    }
}
```