UNIVERSIDAD TÉCNICA FEDERICO SANTA MÁRIA
DEPARTAMENTO DE INFORMÁTICA
CAMPUS SSJJ

Sistemas Operativos,
Pauta Certamen #2, 01/2019
Santiago, 31.07.2019

1. [30% ~ 3% c/u] Conteste si las siguientes preguntas son verdaderas o falsas. En caso que sea falsa, justifique *brevemente*. **Justificación incorrecta no suma puntaje**.

a) Todo proceso está compuesto únicamente por código, datos y stack.
Falso: Y heap.

b) El stack de un proceso es utilizado para almacenar, entre otras cosas, variables globales.
Falso: Las variables globales se almacenan en los datos.

c) La PCB solo guarda información de estado y de registros de apoyo.
Falso: Guarda, además, información de planificación, contabilidad, etc.

d) El itinerador de corto plazo controla el grado de multiprogramación.
Falso: El que control el grado es el itinerador de largo plazo.

e) Hebras de un mismo proceso comparten solamente código y recursos.
Falso: Y datos.

f) Una hebra cambia del estado *running* a *waiting* al ejecutar un **yield**().
Falso: Al ejecutar un join(). La función yield() la deja en el estado ready.

g) Una condición de carrera se produce únicamente en programas multi hebras.
Falso: Programas en ejecución con solo una hebra principal también la presentan.

h) La exclusión mutua satisface el criterio de vivacidad.
Falso: Satisface el criterio de seguridad.

i) Los SO's modernos deben resolver por si mismos los problemas de sincronización.
Falso: Los programadores deben hacerlo, el SO solo entrega las herramientas.

j) Liveness es una propiedad de los SO modernos que asegura el progreso de los procesos.
Verdadero.

2. [20% ~ 4% c/u] **Preguntas con respuestas cortas**:

a) Identifique 3 eventos que pueden ocurrir mientras un proceso se está ejecutando y mencione en qué cola queda asignado:

Respuesta: Pueden ser:

- Solicitar E/S y ser asignado a las colas de espera.
- Crear un nuevo proceso y esperar.
- Ser desalojado como resultado de una interrupción y volver a la cola ready.

b) Identifique 3 beneficios que entrega la comunicación entre procesos cooperativos.

Respuesta:

- Compartir información.
- Cálculos más rápidos.
- Modularidad y diseño.
- Conveniencia en la programación.

c) ¿Cuál es la diferencia entre paralelismo y concurrencia?

Respuesta: El paralelismo ejecuta más de una tarea simultáneamente, pero requiere de una arquitectura multicore. Concurrencia permite que más de una tarea en ejecución avance.

d) Defina brevemente lo que es una condición de carrera.

Respuesta: Condición que se produce cuando el comportamiento de la ejecución de un programa depende de la forma en que se intercalan las operaciones que se deben realizar.

e) Toda solución a un programa de sincronización debe cumplir con:

Respuesta:

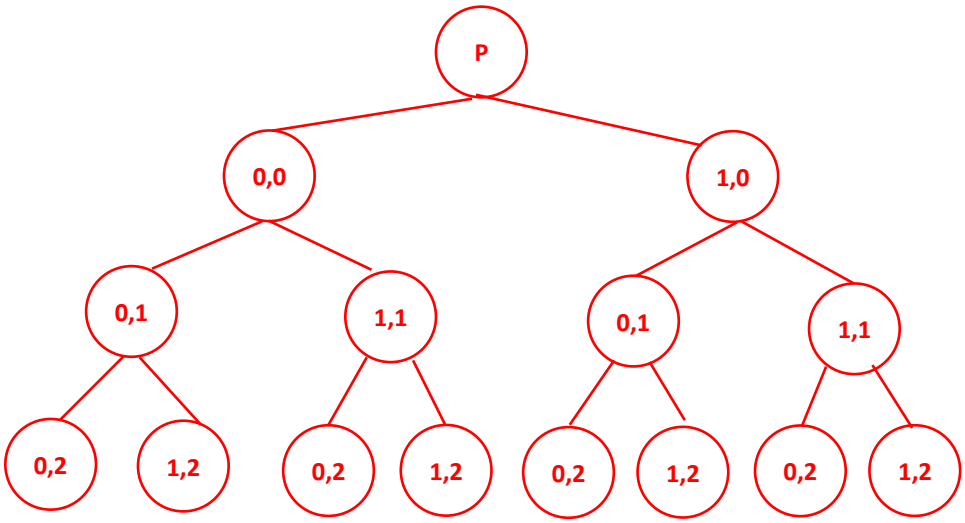
- Exclusión mutua (seguridad).
- Progreso (vivacidad).
- Espera acotada (vivacidad).

3. [25%] **Procesos**: El siguiente código ha sido compilado y ejecutado correctamente. Dibuje la jerarquía de procesos resultantes:

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

#define x 2
#define y 3

int main (int argc, char *argv[]) {
	int u, v;
	pid_t pid;

	for (v=0; v<y; v++){
		for (u=0; u<x; u++){
			pid = fork();
			if (pid==0)
				break;
		}
		if (pid!=0)
			break;
	}
	return 0;
}
```

Para que sea más ordenado, indique el valor de u y v de cuando fue creado cada proceso.

Respuesta: Considere el par u, v para el diagrama:



4. [25%] **Sincronización**: Durante la semana sansana se realizan visitas guiadas en grupos de 5 personas a la universidad. Asuma que solo un grupo entra a la universidad por vez. El visitante que llega debe esperar que se junten las 5 personas y tampoco puede retirarse si el grupo no está completo. Ningún visitante puede entrar si el grupo anterior no ha salido completamente.

Considere que cada visitante es una hebra que ejecuta los siguientes métodos:

```
Visita () {
        Entrar();
        Salir();
}
```

Se pide construir la clase Usm con sus variables de estado y los códigos seguros para las funciones públicas indicadas.

Respuesta: Una posible solución:

```
const visitas_max = 5;
class Usm {
        private:
                Lock lock;
                CV salir, entrar;
                int visitas;
                bool entrando;
        public:
                void Entrar();
                void Salir();
                Usm() {
                        visitas = 0;
                        entrando = 1;
                }
}
Void Usm:: Entrar() {
                lock.acquire();
                while(!(entrando && visitas < visitas_max))
                        salir.wait();
                visitas++;
                if (visitas == visitas_max) {
                        entrar.broadcast();
                        entrando = 0;
                }
                lock.release();
}
```

```
Void Usm:: Salir() {
        lock.acquire();
        while(entrando)
                entrar.wait();
        visitas--;
        if (visitas == 0) {
                entrando = 1;
                salir.broadcast()
        }
        lock.release();
}
```