UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
CAMPUS SAN JOAQUÍN

Sistemas Operativos,
Pauta Certamen #1, Primer Semestre de 2024.

1. Verdadero o Falso (20%)

Conteste si las siguientes afirmaciones son verdaderas o falsas. En caso de ser falsas justifique brevemente. Justificación incorrecta no suma puntaje.

(a) Instrucciones de máquina privilegiadas no requieren soporte hardware.
Falso: Requieren el bit de modo para cambiar entre modo kernel y usuario.

(b) El problema de HAL es que limita a los sistemas a depender del hardware de la máquina.
Falso: Todo lo contrario, un buen diseño de HAL permite a los sistemas ser independientes del hardware.

(c) El kernel contiene la mayoría de los programas de sistema de un sistema operativo.
Falso: Los programas de sistema no son parte del kernel, solo utilizan los servicios de este.

(d) En un SO monolítico un desarrollador puede implementar sus propias llamadas al sistema y utilizarlas.
Falso: Las llamadas se implementan en el kernel, por lo que un desarrollador no puede hacerlo.

(e) La desventaja del micro kernel es que es inseguro, ya que es propenso a uso mal intencionado.
Falso: Todo lo contrario, una ventaja de micro kernel es la seguridad. Desventaja → sobre trabajo.

(f) Un proceso ha ejecutado fork(); La región del Heap es compartida entre padre e hijo.
Falso: Padre e hijo no comparten región en memoria solo por el uso del fork();

(g) El uso de fork() se considera interrupción de tipo E/S y se clasifica como asíncrono.
Falso: El uso de fork() es una llamada al sistema y es síncrona;

(h) Un proceso ejecuta exec(); sin fork(); Se reemplaza el pid, la data, el stack y el heap.
Falso: El pid no cambia, todo el resto sí.

(i) Lo primero al iniciar un computador es que el bootloader cargue el kernel del sistema en memoria.
Falso: Lo primero es que la BIOS haga la revisión del hardware y luego cargue el bootloader.

(j) El diseño y construcción de un SO depende exclusivamente de los mecanismos de implementación.
Falso: El diseño y construcción de un SO depende de los objetivos que persiga.

Evaluación: Cada pregunta vale 2 puntos. 10 preguntas suman 20 puntos en total

- Justificación incorrecta → -2 puntos por pregunta.
- Respuesta incorrecta → -2 puntos por pregunta

2. Preguntas Cortas (20%)

Conteste brevemente las siguientes preguntas.

(a) Regalo: ¿Cuáles son los 3 principales objetivos de un sistema operativo?

Respuesta: Los 3 objetivos principales de un sistema operativo son:
- Proveer un entorno de ejecución de programas de forma concurrente o paralela.
- Asignar de forma justa y eficiente el hardware del sistema.
- Controlas operaciones E/S y prevenir y gestionar errores.

(b) ¿Cuáles son los servicios sugeridos de mantener en el kernel en la estructura de micro kernel de un sistema operativo? Describa brevemente cada uno.

Respuesta: Los grupos de servicios sugeridos son:
- IPC: Comunicación entre proceso. Por ejemplo, paso de mensajes y memoria compartida.
- Asignación del Procesador: Relacionado con la toma de decisiones para asignarlo
- Gestión de Memoria Principal: Mecanismos para asignar y utilizar la memoria principal.

(c) Indique 4 componentes de hardware que forman parte de la jerarquía del hardware, explique los criterios por los cuáles se clasifican y mencione como se relacionan con el objetivo principal del sistema operativo.

Respuesta: Los componentes de hardware son: Disco duro, memoria principal, memoria caché y registro. Los criterios de clasificación son:
- Volatilidad
- Costo
- Tamaño
- Velocidad

Los componentes mencionados corresponden al camino que deben recorrer los programas para ser ejecutados, siendo la gestión de los mismos un factor crítico para el objetivo principal del sistema operativo que es proveer el entorno de ejecución de programas.

(d) En Unix, desde una ventana del terminal (shell/consola), se puede ejecutar $ date, comando que muestra la fecha actual, pero si se ejecuta $ exec date, no se muestra nada y desaparece el terminal ¿por qué?

Respuesta: La ejecución de $ date crea otro proceso para correr el programa date, que al terminar vuelve el control al terminar. Cuando se ejecuta $ exec date, se reemplaza el proceso del terminal por el programa date, de esta forma, al terminar, no hay terminal esperando por él.

Evaluación: Cada pregunta vale 5 puntos. 4 preguntas suman 20 puntos en total
- Respuesta incorrecta → -5 puntos por pregunta

3. Historia y Rendimiento (20%)

En la siguiente tabla se presentan los resultados de ejecutar el mismo programa en dos arquitecturas diferentes y las características de las mismas:

Arquitectura	Instrucciones	CPI	Tasa en GHz (10^9)	E/S [s]	Otras Esperas [s]
ARM	700.000	2,8	3,0	1	1,5
MIPS	1.100.000	1,5	2,1	0,03	0,01

(a) Determine la cantidad de ciclos necesarios para ejecutar el programa en cada arquitectura.
Respuesta:

$$CPI_{ARM} = \frac{\#ciclos}{\#instrucciones} \rightarrow \#ciclos_{ARM} = 2{,}8 \times 7 \times 10^5 \approx 1{,}96 \times 10^6$$

$$CPI_{MIPS} = \frac{\#ciclos}{\#instrucciones} \rightarrow \#ciclos_{MIPS} = 1{,}5 \times 1{,}1 \times 10^6 \approx 1{,}65 \times 10^6$$

(b) Determine el tiempo de uso de CPU para ambas arquitecturas.
Respuesta: El tiempo de uso de CPU viene determinado por el CPI calculado en el punto (a).

$$ARM \rightarrow T_{CPU} = \frac{\#ciclos}{Tasa} \rightarrow \frac{1{,}96 \times 10^6}{3{,}0 \times 10^9} \approx 6{,}53 \times 10^{-4}[s]$$

$$MIPS \rightarrow T_{CPU} = \frac{\#ciclos}{Tasa} \rightarrow \frac{1{,}65 \times 10^6}{2{,}1 \times 10^9} \approx 7{,}9 \times 10^{-4}[s]$$

(c) ¿Qué sistema tiene mejor rendimiento? ¿Por cuántas veces más que el otro?
Respuesta: Los rendimientos vienen dados por:

$$R_{ARM} = \frac{1}{T_{CPU} + ES + Otras} = \frac{1}{6{,}53 \times 10^{-4} + 1 + 1{,}5} = \frac{1}{2{,}500653}$$

$$R_{MIPS} = \frac{1}{T_{CPU} + ES + Otras} = \frac{1}{7{,}9 \times 10^{-4} + 0{,}03 + 0{,}01} = \frac{1}{0{,}04079}$$

$$R_{ARM} = \frac{1}{2{,}500653}, R_{MIPS} = \frac{1}{0{,}04079} \rightarrow \frac{R_{MIPS}}{R_{ARM}} = \frac{2{,}500653}{0{,}04079} \approx 61{,}3$$

Evaluación: a y b valen 6 puntos cada una y c 8 puntos.

- 1 error → -- la mitad de los puntos.
- Más de 1 error → -- todos los puntos.

4. Creación de Procesos (20%)

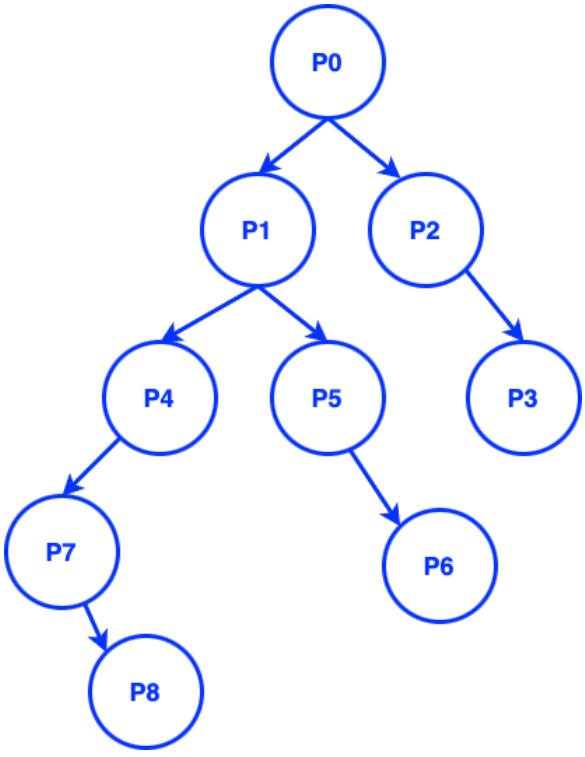
Considere el siguiente extracto de código de un programa que fue compilado sin problemas:

```
…
int main (int argc, char *argv[]) {
       int i, u;
       u = 0;
       for(i=0; i<2; u = fork()||fork()){
           if (u==0){
               i = i + 1;
           }
           else {
               i = i + 2;
           }
       }
       return 0;
}
```

Nota: Los operadores && y || corresponden a las operaciones lógicas AND y OR respectivamente. La asociatividad es de izquierda a derecha y && tiene mayor precedencia que ||. Ambas operaciones evalúan el operando de la izquierda y la evaluación del segundo dependerá exclusivamente del valor de verdad que se obtiene a partir del primero.

Dibuje la jerarquía de procesos resultantes. ¿Cuántos procesos hijos se crearon?

Respuesta: Se crearon 8 hijos. La jerarquía de procesos:



Evaluación: 2 puntos cada proceso. 2 puntos total de hijos

- 1 error → -- la mitad de los puntos.
- Más de 1 error → -- todos los puntos.

5. Comunicación y Rendimiento (20%)

Considere que tiene un programa en C que tiene dos grandes matrices de tamaño MxN. Se requiere construir una solución que permita calcular la suma total de todos los elementos de ambas matrices.

(a) Utilizando solo un proceso hijo y pipes construya una solución que permita explotar una arquitectura de varios núcleos de ejecución de instrucciones.

Respuesta: Una posible solución: El padre calcula el resultado de la suma de la primera matriz y queda a la espera que el hijo le envíe, a través del pipe, el resultado de la suma de la segunda matriz.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int A[M][N] = {…};
int B[M][N] = {…};

int main(int argc, char *argv[])
{
  int fd[2];
  int result, suma, i, j;
  suma = 0;
  pipe(fd);

  if (fork() > 0) {
    for(i=0;i<M;i=i+1)
        for(j=0;j<N;j=j+1)
            suma = suma + A[i][j];
    close(fd[1]);
    read(fd[0], &result, 4);
    printf("The final result is %d\n", result+suma);
    close(fd[0]);
  }
else {
    for(i=0;i<M;i=i+1)
        for(j=0;j<N;j=j+1)
            suma = suma + B[i][j];
    close(fd[0]);
    write(fd[1], &suma, 4);
    close(fd[1]);
}

return 0;
}
```

(b) Si el tiempo de acceso a cada elemento de la matriz y la operación suma es de 1 [ns] realice una estimación de cuánto mejoró el rendimiento con la implementación del punto (a). ¿Podríamos reducir el tiempo aún más? ¿Cómo? ¿Cuál es el límite?

Respuesta: Despreciando todos los tiempos y solo dando foco al acceso a las matrices, el tiempo de ejecución del programa sin dividirlo es de MxNx2. Al dividir con un hijo y utilizar pipes, en una arquitectura don dos núcleos el tiempo podría disminuir a la mitad.

Podríamos mejorar más aún este tiempo dividiendo el recorrido de las matrices en más hijos. Por ejemplo, ocupar al padre más 3 hijos y hacer que estos recorran la mitad de las matrices. Esto podría reducir el tiempo inicial hasta en un cuarto. Podríamos plantear una estrategia de seguir dividiendo el recorrido de las matrices pero dependerá de la cantidad de núcleos que tenga el procesador.

Evaluación: Pregunta (a) 15 puntos. (b) 5 puntos

- 1 error → -- la mitad de los puntos.
- Más de 1 error → -- todos los puntos.