UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
CAMPUS SAN JOAQUÍN

Sistemas Operativos, Pauta Certamen #1
Segundo Semestre de 2023, Santiago.

1. Historia, Hardware y Rendimiento (25%)

Parte 1 – 12%

Se ha instalado Arch Linux en un computador que cuenta con una tasa de reloj de 2Ghz. La ejecución de un programa X (400 mil millones de instrucciones) demora 13 minutos y 20 segundos.

(a) Determine el CPI del programa X.
Respuesta:

$$CPI_X = \frac{T_{CPU} * Tasa}{Ins} = \frac{800\ [s] * 2 * 10^9\ [s]^{-1}}{4 * 10^{11}} = 4$$

(b) Calcule el tiempo de CPU para un programa Y que tiene un 20% más de instrucciones que X y el CPI corresponde al 75% del calculado en el punto (a).
Respuesta:

$$T_{CPU} = \frac{CPI * Ins}{Tasa} = \frac{3 * 1{,}2 * 4 * 10^{11}}{2 * 10^9} = 720\ [s]$$

(c) Si otro computador obtiene un CPI de 3,5 para el programa X, ¿cuál es más rápido y por cuánto?
Respuesta: Si se ejecutan a misma tasa de reloj y tienen la misma cantidad de instrucciones:

$$\frac{T_{CPU2}}{T_{CPU1}} = \frac{CPI_1}{CPI_2} = \frac{4}{3{,}5} \approx 1{,}14$$

(d) En el contexto de esta pregunta: ¿cómo podemos mejorar el rendimiento del sistema al ejecutar X?
Respuesta: Con la poca información que se maneja:

- Aumentar la frecuencia del procesador.
- Reducir la cantidad de ciclos necesarios para ejecutar X.

Parte 2 – 13%

Se requiere ejecutar un programa que tiene el siguiente código en C (asuma que el programa no hace nada más y que las definiciones de variables, arreglos y retornos no son parte del análisis solicitado) en un procesador de periodo 3 [µs] y que ejecuta instrucciones en un ciclo de reloj.

```
…
for (i=0; i<1000; i=i+1)
  for (j=0; j<1000; j=j+1)
    uwu[i][j] = uwu[i][j] + 1;
…
```

Durante la ejecución del extracto anterior, se producen 10 interrupciones, en dónde al SO le toma 1 [s] realizar el cambio de contexto y el proceso debe esperar 2 [s] cada vez para volver a tomar el procesador.

Determine el tiempo de ejecución.
Respuesta: El ciclo interno requiere 2 accesos a memoria, 2 operaciones aritméticas y 2 direccionamientos (para mantenerse dentro del ciclo y salir). El ciclo externo requiere 2 operaciones aritméticas (una de ellas para resetear j) y 2 de tipo direccionamiento.

$$T_{Ej} = \big((2 + 2 + 6000) * 1000\big) * 3 * 10^{-6} + 20 * 1 + 2 * 10 = 58{,}012\ [s]$$

Evaluación:

Parte 1: 4% cada una.

- 1 error → -2 puntos.
- + de 1 error → -4 puntos.

Parte 2: 13%.

- 1 error → -3 puntos.
- 2 errores → -8 puntos
- + de 2 errores → -13 puntos.

2. Organización, Diseño y Rendimiento (25%)

Parte 1 – 10%

El tiempo de ejecución de un programa Z en una nueva distribución de Unix es de 50 [s]. Considere que la ejecución solo se vio interrumpida por las llamadas al sistema generadas por él.

(a) Indique las actividades generales que se deben realizar para resolver una llamada al sistema.
Respuesta: Las actividades:

- Invocación de la llamada desde el programa.
- Invocación al kernel handler desde la librería.
- Localiza argumentos y los copia en una región accesible.
- Validar los valores para evitar errores.
- Invoca a la rutina resolutora.
- Se ejecuta la rutina.
- Se copian los resultados en una región de memoria para el usuario.
- Se reanuda la ejecución.

(b) Si a cada una de las actividades del punto (a) le toma 1[ms] llevarse a cabo y se sabe que el 20% del tiempo de ejecución de Z correspondió a gestionar estos eventos, determine la cantidad de llamadas al sistema que fueron utilizadas.
Respuesta: Del punto anterior se desprende que se requieren 8 [ms] en cada invocación.

$$Total_{llamadas} = \frac{10\ [s]}{8 * 10^{-3}\ [s]} = 1250$$

Parte 2 – 15%

(a) ¿De qué factores depende la portabilidad en un sistema operativo como Windows? Explíquelos.
Respuesta: Windows es un sistema operativo monolítico:

- HAL: Código y operaciones que permiten abstraerse de la arquitectura de una máquina.
- Drivers: Identificar todos los drivers de E/S y si no descargarlos desde internet.

(b) Al momento de virtualizar ¿cómo aseguramos la seguridad e integridad del sistema operativo host? ¿En qué puede afectar esto?
Respuesta: El sistema operativo visita se ejecuta siempre en modo usuario del host. Si se requiere ejecutar instrucciones privilegiadas es la VMM quien solicita el apoyo del host y retorna los resultado. Esto impacta directamente en el rendimiento.

(c) ¿Por qué se dice que los Sistemas Operativos son conducidos por interrupciones? ¿Impacta esto en el rendimiento?
Respuesta: Los sistemas operativos deben definir las interrupciones que soportan e implementar las rutinas que las resuelven. Estos eventos van desde errores hasta implementación de los mismos servicios del sistema, por lo que son un evento común que interrumpe el flujo normal de ejecución de los programas. Puede impactar en el rendimiento debido a los excesivos cambios de contexto que se podrían producir, pero es algo con lo que debemos vivir.

Evaluación:

Parte 1: 5% cada una.
- 1 error → -2 puntos.
- + de 1 error → -5 puntos.

Parte 2: 5% cada una.
- 1 error → -2 puntos.
- + de 1 error → -5 puntos.

3. Procesos - Creación (25%)

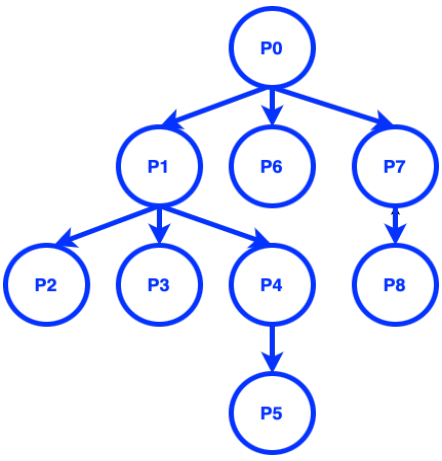
Parte 1 – 15%

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

```
…
int main (int argc, char *argv[]) {
        if (fork() || fork()){
                if (fork() && !fork()){
                    fork();
                }
        }
        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.
Respuesta: La jerarquía resultante:



Parte 2 – 10%

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

```
…
int main(int argc, char *argv[])
{
  int a = 5, b = 10, c = 20, d = 30, ferxxxo;
  ferxxxo = fork() ? (fork() ? a : (fork() ? b : c)) : d;
  printf("%d\n", ferxxxo);
  return 0;
}
```

Explique brevemente el comportamiento de los valores de salidas e indique algunas posibles opciones.
Respuesta: Dos posibles salidas: 30 5 10 20 y 5 30 10 20. Esto se debe a la forma de operación de la combinación ternaria. Al ejecutar el primer fork() si no se cumple inmediatamente es 30, caso contrario entra y vuelve a ejecutar fork(), si se cumple muestra 5, dado esto siempre se mostrará alguno de los dos en primer lugar y luego viene la segunda evaluación que siempre seguirá el orden 10 y 20.

Evaluación:

Parte 1: 15%.
- 1 error → -5 puntos.
- 2 errores → -10 puntos
- + de 2 errores → -15 puntos.

Parte 2: 10% cada una.
- 1 error → -5 puntos.
- + de 1 error → -10 puntos.

4. Procesos - Comunicación (25%)

Considere que se tiene un programa que tiene dos arreglos de un tamaño N bastante grande. Se desea calcular la suma total de todos los elementos de cada arreglo. Para no hacerlo de forma secuencial y tener la capacidad de explotar una arquitectura de múltiples procesadores se solicita programar la solución que permita lograr el objetivo indicado. El padre es quién debe mostrar el resultado final por pantalla.

(a) Utilice pipes para poder comunicar y la menor cantidad de fork's y wait's posibles.

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

#DEFINE N = Too big.
int  A[N] = {1,2,3, …}, B[N] = {4,5,6, …};

int main(int argc, char *argv[])
{
    …
    return 0;
}
```

Respuesta: Una posible solución: Se requieren dos pipes y dos hijos para recorrer y sumar los arreglos. Ejemplo con arreglos de tamaño 3, se extiende para N fácilmente.

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

int  a[3] = {1,2,3}, b[3] = {4,5,6};

int main(int argc, char *argv[])
{
  int fd[2];
  int result;

  pipe(fd);

  if (fork() > 0) {
    close(fd[1]);
    read(fd[0], &result, 4);
    printf("The final result is %d\n", result);
    close(fd[0]);
  }
  else {
    int sum = 0, i;
    int fd2[2];
    pipe(fd2);
    if (fork() > 0) {
       for (i = 0; i < 3; i++){
        sum = sum + a[i];
      }
      close (fd2[1]);
      read(fd2[0], &result, 4);
      sum = sum + result;
      close (fd[0]);
      write(fd[1], &sum, 4);
      close(fd[1]);
    }
    else {
      for (i = 0; i < 3; i++){
        sum = sum + b[i];
      }
      close (fd2[0]);
      write(fd2[1], &sum, 4);
      close(fd2[1]);
    }
  }
  return 0;
}
```

(b) ¿Qué ocurre con la estrategia si quiero agregar más arreglos? ¿Hay una mejor opción?
Respuesta: Siguiendo esta estrategia se requieren tantos pipes como arreglos se deban recorrer, lo cual se vuelve un poco ineficiente. Se debe buscar y utilizar un mejor mecanismo de comunicación y sincronización, es más, el uso de hebras se vuelve mucho más eficiente para este tipo de tareas.

Evaluación:

Parte (a): 20%.

- 1 error → -5 puntos.
- 2 errores → -10 puntos
- + de 2 errores → -20 puntos.

Parte (b): 5% cada una.

- 1 error → -5 puntos.