

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

Midterm

IIC 2333 — Sistemas Operativos y Redes
26 Abril 2023

Duración: 2 horas

Parte 1: Procesos y Syscalls

Respecto a la materia de Procesos, responda de manera breve y precisa las siguientes preguntas:

1.1 **[4p]** Responda V o F, Justificando las falsas.

(a) Un proceso esta compuesto por el código y los recursos que utiliza (Ej:Memoria).

- **Respuesta:** Verdadero

(b) La Syscall `wait()` espera el término de un proceso hijo.

- **Respuesta:** Verdadero

(c) Syscall `exit()` termina el proceso con un código de retorno dado, y lo entrega al padre.

- **Respuesta:** Verdadero

(d) El costo de un cambio de contexto en un procesador se debe principalmente a la necesidad de reemplazar los datos *Process Control Block*.

- **Respuesta:** Verdadero

1.2 **[1p]** Explique que es un proceso Huérfano.

Respuesta: Es un proceso el cual su padre terminó sin recibir su código de salida.

1.3 **[10p]** Respecto a la materia de syscalls y procesos, responde las siguientes preguntas a partir del siguiente código.

```c
void signal_handler_alarm() {

    for (int i = 0; i < 10; i++) {
        kill(pid[i], SIGKILL);
    }
}

int main() {

    signal(SIGALRM, signal_handler_alarm);

    for (int i = 0; i < 10; i++) {

        pid[i] = fork();
        if (pid[i] == -1) {
            perror("fork");
            return EXIT_FAILURE;
```

```
18
19          } else if (pid[i] == 0 && i <= 4) {
20              sleep(5);
21              char *argv[] = {"/bin/echo", "Hello World!", NULL};
22              execv(argv[0], argv);
23              return EXIT_SUCCESS;
24
25          } else if (pid[i] == 0 && i > 4) {
26
27              execl("/bin/pwd", "/bin/pwd", (char *) NULL);
28              return EXIT_SUCCESS;
29          }
30      }
31
32      alarm(3); // Establece un temporizador de 3 segundos.
33
34      for (int i = 0; i < 10; i++) {
35        int status;
36        waitpid(pid[i], &status, 0);
37        printf("Proceso hijo %d terminado con status %d\n", pid[i], status);
38      }
39      printf("Proceso padre terminado \n");
40      return 0;
41  }
```

(a) **[3p]** A partir del código, identifique y explique para que sirven las syscalls y las señales involucradas.

Respuesta:

- **0.5p**. fork: Genera un duplicado del proceso actual, es decir, es una copia del proceso padre. La función crea un proceso hijo a partir del proceso padre y devuelve el PID (Identificador de proceso) del hijo.
- **0.5p**. waitpid: Se utiliza para que el proceso padre espere a que sus procesos hijos terminen su ejecución. Esta syscall se encuentra en el segundo bucle `for` y espera a que cada uno de los procesos hijos termine para continuar con la ejecución del proceso padre.
- **0.5p**. exec (execv y execl): Reemplaza el proceso por otro y comúnmente utilizada por los programas para ejecutar otros programas en el sistema operativo. Además, incluye varias funciones con diferentes nombres y argumentos, en el código se encuentran execl() y execv().
- **0.5p**. kill: Envía una señal a un proceso.
- **0.5p**. SIGALRM: Señal que se envía luego de un tiempo determinado.
- **0.5p**. SIGKILL: Señal de termino forzado.

(b) **[2p]** Si eliminamos las líneas 34 al 38, ¿qué pasaría con el flujo del programa?¿por qué es importante el `waitpid` en el programa?.

Respuesta:

Al eliminar las líneas de código que incluyen la función waitpid(), se produce un efecto indeseado en el programa, lo que resulta en procesos huérfanos. La función waitpid() se utiliza en el programa para que el proceso padre espere a que los procesos hijos terminen de ejecutarse antes de continuar. Al eliminar esta función, el proceso padre no esperará a los procesos hijos y terminará de ejecutarse sin esperarlos. Esto provoca que los procesos hijos queden sin un proceso padre y, por lo tanto, se conviertan en procesos huérfanos.

- **1p**. Por identificar que se generan procesos huerfanos.
- **1p**. Por explicación correcta.

(c) **[2p]** Se obtiene el siguiente output, ¿por qué los primeros cinco procesos terminan con status 9?.

```
Proceso hijo 343 terminado con status 9
Proceso hijo 344 terminado con status 9
Proceso hijo 345 terminado con status 9
Proceso hijo 346 terminado con status 9
Proceso hijo 347 terminado con status 9
Proceso hijo 348 terminado con status 0
Proceso hijo 349 terminado con status 0
Proceso hijo 350 terminado con status 0
Proceso hijo 351 terminado con status 0
Proceso hijo 352 terminado con status 0
Proceso padre terminado
```

Respuesta:

- **0.5p**. Por identificar que `alarm(3)` propaga la señal `SIGALRM` luego de 3 segundos.
- **0.5p**. Por identificar `SIGALRM` que es capturada por `signal(SIGALRM, signal_handler_alarm)`.
- **1p**. Por identificar debido a la señal `SIGKILL`, los procesos hijos tienen código de salida 9.

(d) **[3p]** Considerando el código y el item 1.3, te comentan que no puedes modificar la función `void signal_handler_alarm()` ni evitar la propagación de la alarma. Proponga una manera de asegurar que el status de los primeros cinco procesos sea 0.

Respuestas posibles (3p):

1. Eliminar `sleep(5)`.
2. Mover `sleep(5)` después del `execv()`
3. Aumentar el tiempo de `alarm(3)` a 6 o más.
4. Mover `alarm(3)`, que puede ser en:
   - después del `sleep(5)` (en la línea 20 - 22)
   - después del `else if` (en la línea 25 - 27)
   - después del `waitpid` (a la línea 36)

Parte 2: Threads y Sincronización

2.1 **[5p]** Respecto a la materia de Threads y Sincronización, responda V o F, justificando las falsas de manera breve:

(a) Un cambio de user mode a kernel mode requiere de un cambio de contexto.

- **Respuesta:** Falso

(b) La función `yield()` permite entregar el turno a otros threads.

- **Respuesta:** Verdadero

(c) Un proceso de nivel usuario puede escribir en la memoria de un proceso del kernel.

- **Respuesta:** Falso

(d) Un sistema multicore no puede implementar multitasking, pero un sistema single-core sí.

- **Respuesta:** Falso

(e) Cuando implemento los threads en el kernel tengo escalabilidad límitada por el SO, dado que no me permite más de una cierta cantidad establecida de threads.

- **Respuesta:** Verdadero

2.2 **[5p]** La próxima vez que vaya a una tienda (farmacia, banco, etc) y tenga que sacar un número para solicitar atención, piense que está ejecutando un protocolo de exclusión mutua entre múltiples *threads*. Suponga que existe solo un puesto de atención (1 CPU), y múltiples clientes (*threads* que solicitan atención y para ello escogen un número para ser atendidos en orden creciente de ese número. Más aún, no hay una máquina común (variable compartida) para obtener el siguiente número de atención, pero es posible observar el número de todos los demás clientes.

Para `n` *threads* se tiene un arreglo booleano `escogiendo[0:n-1]` inicializado en `FALSE`, y un arreglo de enteros `numero[0:n-1]` inicializado en 0. Una característica interesante del siguiente algoritmo es que el *thread* i puede leer cualquier posición de cada arreglo, pero sólo escribe en la posición i.

```
1  /** Codigo para thread i. Los arreglos 'escogiendo' y 'numero' son compartidos. **/
2  do {
3    escogiendo[i] = TRUE;
4    numero[i] = 1 + max(numero[0], numero[1], ..., numero[n-1]);
5    escogiendo[i] = FALSE;
6    for (j=0; j<n; j++) {
7      while(escogiendo[j] == FALSE);
8      if(numero[j] != 0 && (numero[j] < numero[i]) )
9        while(numero[j] > 0);
10   }
11   /** SECCION CRITICA **/
12   numero[i] = 0;
13   /** FIN SECCION CRITICA **/
14 } while(true);
```

(a) Argumente si este código cumple o no con las propiedades de solución al problema de la **sección crítica**.
Respuesta:

- **[2p]**. Exclusion mutua: Cada thread se bloquea en la línea 7 al consultar por su propio estado escogiendo, y no puede liberarse. Hay exclusión mutua porque no hay más de un thread en su sección crítica.
- **[1.5p]**. Progreso: No hay progreso. Si más de un thread desea entrar, ninguno puede hacerlo porque se bloquean.
- **[1.5p]**. Espera acotada: No hay espera acotada. Ya que nadie puede entrar, el tiempo que deben esperar para entrar es infinito.

Parte 3: Scheduling

3.1 **[5p]** Respecto a la materia de scheduling, responda V o F, justificando las falsas de manera breve:

(a) La acción de *Swap* es sacar la memoria del proceso de la memoria principal y llevarlos al espacio de swap, y viceversa.

- **Respuesta:** Verdadero

(b) CPU Intensive son pocas ráfagas largas de cómputo en la CPU.

- **Respuesta:** Falso

(c) Multitasking es asignar tiempo a múltiples procesos.

- **Respuesta:** Verdadero

(d) La CPU (single core) puede atender múltiples procesos a la vez (mismo instante de tiempo).

- **Respuesta:** Falso

(e) Un scheduler expropiativo, permite que el proceso ejecute hasta que termine.

- **Respuesta:** Falso

3.2 **[10p]** Implementación scheduler

(a) **[3p]** ¿Por qué un scheduler FCFS (First-Come First-Serve) no es apropiado para procesos interactivos? Mencione uno que sí lo sea, y justifique brevemente.

Respuesta:

- **3p**. FCFS no es apropiado para procesos interactivos, ya que este busca minimizar el turn around time de los procesos. Aquellos procesos interactuan con el usuario y necesitan un response time rápido para que el usuario no quede esperando una respuesta al proceso.

(b) **[7p]** Se tiene un nuevo scheduler y le piden que lo ponga a prueba. Las características del scheduler son las siguientes:

- Scheduler cuenta con 2 colas, una de prioridad alta (cola `High`) y otra de prioridad baja (cola `Low`). Los procesos en la cola `High` siempre tienen prioridad sobre la cola `Low`.
- Los procesos entran al sistema en estado `READY` y en la cola de prioridad alta (cola `High`).
- El scheduler usa un sistema Round-Robin en ambas colas de quantum = 2.
- Un proceso sale de la CPU si la cede o si se acaba su quantum. Si estos pasan en el mismo tiempo, entonces se considera que el proceso fue interrumpido.
- Si un proceso en cola de prioridad alta (cola `High`) termina su quantum, baja a una cola de prioridad baja (cola `Low`). Si esta en una cola de prioridad baja (cola `Low`), se mantiene en esta y se reinicia su quantum.
- Los procesos solo reinician su quantum al cambiar de cola de prioridad (de cola `High` a cola `Low` o `Low` a `High`).
- La prioridad dentro de una cola esta determinada por FIFO (FCFS).
- Si un proceso pasa más de 3 segundos desde la última vez que salió de la CPU y esta en la cola `Low`, se debe subir al final de la cola `High`.

Al scheduler se le asignan los siguientes procesos:

Nombre	Tiempo Ingreso	Total CPU Use	CPU Burst	I/O Wait
process1	1	5	3	3
process2	8	2	2	0
process3	0	5	2	2

Tiene la siguiente tabla con información incompleta del scheduler:

Nombre	0	1	2	3	4	5	6	7
process1		Re/H	Ru	Ru	Re/L	Re/L	Ru	W/L
process2								
process3	Ru	Ru	W/L	W/L	Ru	Ru	W/L	W/L

H= Cola High; L= Cola Low; Ru= Running; Re= Ready; W=I/O wait.

a) **[4p]** Indique el estado y la cola en que se encuentran los procesos en el segundo 10.
Respuesta:
- **2p**. Process1: Estado= Ready; Cola= High.
- **1p**. Process2: Estado= Finished; Cola= None.
- **1p**. Process3: Estado Running; Cola= None.

b) **[3p]** Calcule el *turnaround time* promedio de los procesos.
Respuesta:
- **1p**. Process1: 12
- **1p**. Process2: 2
- **1p**. Process3: 11
- **-1p**. Promedio: 8.3333333 (-1 mal calculo del promedio).

Parte 4: Memoria y Archivos

4.1 **[12p]** Memoria

4.1.1) **[2p]** Respecto a la materia de Paginación, responda Verdadero o Falso en las siguientes preguntas, justifique las falsas:

(a) Cuando ocurre un evento de pagefault, el sistema operativo elimina de la tabla de procesos al proceso que lo genero.
- **Respuesta:** Falso

(b) Los algoritmos de reemplazo de páginas buscan minimizar la cantidad de Page Fault y aumentar el hit rate.
- **Respuesta:** Verdadero

4.1.2) **[10p]** Considere un sistema cuyo hardware está diseñado para soportar hasta 8GB de memoria física, y el sistema operativo configurado para manejar páginas de 1KB. El espacio de direcciones virutales utiliza 28 bits.

(a) **[4p]** Indique la cantidad de memoria virtual direccionable, y la cantidad de bits necesarios para el offset, número de página y número de frame.
Respuesta:
- **1p**. |Mem virtual| = 2^{28} = 256 MB
- **1p**. offset: 1KB = 2^{10}. Por lo tanto, necesitamos 10 bits.
- **1p**. VPN: 256MB/1KB = 2^{18}. Por lo tanto, necesitamos 18 bits.
- **1p**. PFN: 8GB/1KB = 2^{23}. Por lo tanto, necesitamos 23 bits.

(b) **[6p]** Diseñe un sistema de direccionamiento de memoria usando paginación multinivel, de manera tal que cada tabla de páginas quepa completamente dentro de una página. Considere que se deben incluir 4 bits de meta datos y que las entradas de la tabla de páginas (PTE) deben tener un tamaño múltiplo de 1B (8 bits). Especifique la cantidad de bits y los tamaños de tabla en cada nivel, e indique la cantidad mínima de memoria física necesaria para poder hacer una traducción. Suponga que las tablas tienen el mismo tamaño de PTE, es decir, el tamaño de PTE del último nivel. En caso de no responder la pregunta anterior, asuma que la cantidad de bits para el PFN y el VPN son 26 y 22 respectivamente.

Respuesta Posible:

Cálculamos el tamaño de entrada:

$$\begin{aligned} |\text{PTE}| &= \left\lceil \frac{|PFN| + |metadata|}{8} \right\rceil B \\ &= \left\lceil \frac{23+4}{8} \right\rceil B \\ &= 4B \end{aligned}$$

Cálculo de máximo de entradas en un nivel:

$$\begin{aligned} \text{Máx entradas} &= \lfloor |\text{Página}| / |\text{PTE}| \rfloor \\ &= \lfloor 1KB/4B \rfloor \\ &= 2^8 \end{aligned}$$

Con esto podemos hacer un esquema de tres niveles utilizando 5, 5 y 8 bits para cada nivel respectivo. Luego, la cantidad mínima para hacer una traducción es $2^5 \cdot 4B + 2^5 \cdot 4B + 2^8 \cdot 4B = 128B + 128B + 1KB$.

- **1.2p**. Cálculo de PTE.
- **1.2p**. Cálculo de máximo de entradas.
- **2.4p**. Proponer un esquema de 3 o más niveles que sumen 18 bits en total.
- **1.2p**. Calcular el tamaño mínimo para la consulta.

4.2 **[8p]** Pregunta Archivos

4.2.1) **[3p]** Describa el funcionamiento de un sistema de archivos, desde el punto de vista de los servicios que ofrece (al usuario), y los servicios que utiliza.

Respuesta:

3p. Los sistemas de archivos permiten al usuario organizar jerarquicamente utilizando Sistema de archivos ofrece una interfaz/servicio para editar/crear/modificar una abstraccion de archivos y directorios. Permite organizar mis archivos proporcionandome una serie de API para realizar las funciones más comunes.

4.2.2) **[3p]** Respecto a la materia de Archivos, responda Verdadero o Falso en las siguientes preguntas, justifique las falsas:

a) El *Volume Control Block* guarda los datos de formato del sistema de archivos.
   - **Respuesta:** Verdadero

b) Los *SSD* tienen latencia rotacional y consumen más energía que un disco convencional.
   - **Respuesta:** Falso, lo SSD consumen menos energia y no tienen latencia rotacional ni seek time.

c) Un archivo es la unidad mínima de almacenamiento para el usuario.
   - **Respuesta:** Verdadero

4.2.3) **[2p]** Describa que es un *symbolic link* y un *hard link*.

Respuesta:

Un hard link es una referencia al archivo, solo se elimina el archivo si se eliminaron todas las referencias. En cambio un symbolic link es un link a nombre de archivo (ln -s).

- **1p**. Describir correctamente *hard link*.
- **1p**. Describir correctamente *symbolic link*.