Procesos y Syscalls

Contenidos

Proceso Huérfano

```c
int main() {
    pid_t pid = fork();

    if (pid < 0) {
        perror("Error al crear proceso");
        exit(1);
    } else if (pid == 0) {
        printf("Soy el hijo con PID: %d\n", getpid());
        printf("Mi padre es: %d\n", getppid());
        sleep(5);
        printf("Ahora el padre es: %d\n", getppid());
    } else {
        printf("Soy el padre con PID: %d\n", getpid());
        sleep(1);
        printf("El padre termina\n");
    }

    return 0;
}

```

Proceso Zombie

```c
int main() {
    pid_t pid = fork();

    if (pid < 0) {
        perror("Error al crear proceso");
        exit(1);
    } else if (pid == 0) {
        printf("Soy el hijo con PID: %d\n", getpid());
        printf("Hijo termina\n");
        exit(0);
    } else {
        printf("Soy el padre con PID: %d\n", getpid());
        sleep(10);
        printf("Padre termina después de 10 segundos\n");
    }

    return 0;
}

```

Syscalls

```c
int main() {
    pid_t pid = fork();

    if (pid < 0) {
        perror("Error en fork");
        exit(1);
    }

    if (pid == 0) {
        printf("Soy el hijo, mi PID es: %d\n", getpid());
        execlp("/bin/sleep", "sleep", "10", NULL);
        perror("Error en exec");
        exit(1);
    } else {
        printf("Soy el padre, mi PID es: %d\n", getpid());
        sleep(3);
        printf("Padre enviando señal para matar al hijo...\n");
        kill(pid, SIGKILL);

        int status;
        waitpid(pid, &status, 0);

        if (WIFSIGNALED(status)) {
            printf("Hijo fue terminado por la señal: %d\n", WTERMSIG(status));
        }

        printf("Padre termina.\n");
    }

    return 0;
}

```

Crea un nuevo proceso hijo duplicando el proceso actual.

Reemplaza el espacio de memoria del proceso actual con un nuevo programa.

Enviar señales a un proceso.

Espera a que uno o más procesos hijos terminen su ejecución.

Ejercicios

Verdadero/Falso

1. Un proceso esta compuesto por el código, datos, stack y los recursos que utiliza (Ej:Memoria).
2. Un proceso finalizará su ejecución siempre cuando reciba una señal enviada con kill().

Respuestas

1. Verdadero
2. Falso. Existen señales que no buscan finalizar el proceso, señales que pueden ser ignoradas o manejadas de tal manera que no finalicen la ejecución.

Respecto a la materia de syscalls y procesos, responde las siguientes preguntas a partir del siguiente código.

1. ¿Cuántos procesos hijos se crean? No considere el proceso principal y se realiza con éxito la ejecución del exec.
2. ¿Se imprime el PID del primer hijo o del segundo hijo?.
3. ¿Qué pasaría si el execlp falla?

```c
int main() {
    pid_t pid_1 = fork();

    if (pid_1 < 0) {
        perror("Error al crear proceso");
        exit(1);
    } else if (pid_1 == 0) {
        sleep(2);
    } else {
        execlp("ls", "ls", NULL);
        pid_t pid_2 = fork();
        pid_t terminated_pid = wait(NULL);
        printf("El proceso hija terminado tiene PID: %d\n", terminated_pid);
    }
    return 0;
}
```

Respuestas

1. Solo 1.
2. No se imprime.
3. Si falla, se ejecutarán las líneas 11, 12 y 13.

Scheduling

Contenidos

Preemptive vs Non-preemptive Scheduling

Característica	Preemptive scheduling algorithms	Non-preemptive scheduling algorithms
Definición	Pueden interrumpir un proceso en ejecución para dar paso a otro.	No pueden interrumpir un proceso; este debe terminar o esperar.
Control del CPU	El sistema operativo tiene control sobre el proceso en ejecución.	El proceso debe ceder el control del CPU voluntariamente.
Ejemplos de Algoritmos	Round Robin	First Come, First Served (FCFS), Shortest Job First (SJF)
Eficiencia en tiempo de respuesta	Generalmente, mejora el tiempo de respuesta para procesos interactivos.	Puede llevar a tiempos de espera más largos, especialmente si un proceso largo se ejecuta primero.
Riesgo de Starvation	Menor riesgo, aunque puede ocurrir con prioridades muy desiguales.	Mayor riesgo, ya que los procesos de menor prioridad pueden no ejecutarse si siempre llegan procesos de mayor prioridad.

Tipos

FIFO

Los procesos se ejecutan en el orden en que llegan.

Shortest Job First (SJF)

Se ejecuta el proceso con el tiempo de ejecución más corto.

Priority Scheduling

Se ejecuta el proceso con la mayor prioridad.

Round Robin (RR)

Cada proceso se ejecuta por un tiempo fijo (quantum).

Multilevel Feedback Queue Scheduling

Usa múltiples colas y mueve procesos a colas de menor prioridad si usan demasiados recursos.

Ejercicio

Verdadero/Falso

1. Un scheduler expropiativo, permite que el proceso ejecute hasta que termine.
2. El scheduler Shortest Job First (SJF) no puede provocar starvation.

Respuestas

1. Falso, un scheduler expropiativo tienen la facultad de expropiarle la CPU a un proceso antes que termine, en caso de que se cumplan ciertos requisitos, tiempo de ejecución, estado de otros procesos etc.
2. Falso. El scheduler SJF puede provocar starvation debido a que nunca dará turno a los procesos que poseen rafagas de CPU-burst largos.

3.3 Para evaluar el rendimiento de los schedulers *Shortest Job First (SJF) y First In, First Out (FIFO)* en la ejecución de procesos con una sola *CPU-Burst*. Se provee la siguiente información.

Proceso	Tiempo inicial(Llegada)	CPU-Burst
P1	0	15
P2	0	5
P3	2	10

Calcular el Response Time & Turnaround Time!

Tarea

Midterm

¿Qué es?

Es una evaluación que abarcará todos los contenidos cubiertos de la parte de sistemas operativos.

IIC2333 - Sistemas Operativos/ 2024-2

Ayudantía Midterm

Contenidos

1. Procesos y Syscalls
Se refiere a la ejecución de programas y las llamadas al sistema que permiten a los procesos interactuar con el sistema operativo (como abrir archivos o solicitar memoria).

2. Scheduling
Se refiere a cómo el sistema operativo asigna el tiempo de CPU entre procesos o hilos para maximizar eficiencia y rendimiento.

3. Threads y Sincronización
Se refiere a la ejecución paralela dentro de un proceso (hilos) y cómo se coordinan para evitar conflictos y garantizar resultados correctos mediante mecanismos de sincronización.

4. Memoria y Archivos
Trata sobre la gestión de la memoria (RAM) y el almacenamiento de archivos, asegurando que los programas tengan acceso eficiente a los recursos que necesitan.

Threads y Sincronización

Sincronización

Race Condition

Cuando dos o más procesos acceden a los mismos datos al mismo tiempo y, debido a esto, el resultado final puede ser inesperado o incorrecto. Esto sucede porque no hay un control adecuado sobre el acceso a esos datos.

Sección Crítica

Exclusión mutua

Solo un proceso accede a la sección crítica a la vez.

Progreso

Los procesos no deben esperar indefinidamente.

Espera Acotada

Solo los procesos interesados participan en la decisión de entrar a la sección crítica.

Ejercicio

2.4 La próxima vez que vaya a la sala de ayuda y tenga que solicitar ayuda a uno de los ayudantes, piense que está ejecutando un protocolo de exclusión mutua entre múltiples *threads*. Suponga que existe solo un ayudante (1 CPU), y `n` estudiantes (*threads*) que desean recibir ayuda.

Para `n` estudiantes, se tiene un arreglo compartido `necesitaAyuda[0:n-1]`, donde `necesitaAyuda[i]` indica si el estudiante `i` esta pidiendo ayuda. Se cuenta además con la variable compartida `turnoActual` el cual indica a que estudiante le toca ser atendido.

Por último, se tiene la función `RecibirAyuda(i)` la cual se utiliza para representar que el estudiante `i` esta siendo atendido, esta función solo debe estar siendo ejecutada por a lo más un *thread*. También, se cuenta con la función `ContinuarTarea(i)` que representa la acción del estudiante que continua realizando su tarea, esta función puede puede ser ejecutada por múltiples threads en paralelo.

Se propone el siguiente código para modelar el protocolo de exclusión mutua (para simplificar consideramos solamente 3 estudiantes):

```c
// variables compartidas
bool necesitaAyuda[3] = {false, false, false};
int turnoActual = 0;
```

```c
/** Codigo para thread i **/
while (true) {
  necesitaAyuda[i] = true;
  turnoActual = (i + 1) % 3;
  while ((necesitaAyuda[(i+1) % 3] || necesitaAyuda[(i+2) % 3]) && turnoActual != i);
  /** Inicio Sección Crítica **/
  RecibirAyuda(i);
  necesitaAyuda[i] = false;
  /** Final Sección Crítica **/
  ContinuarTarea(i);
}
```

A partir del código anterior responda la siguientes preguntas:

(a) **[3p]** Argumente si en la(s) sección(es) críticas se cumplen las propiedades de solución al problema de la **sección crítica**.

Respuesta

Respuesta:

- **1 pts**. Exclusión mutua **no se cumple**, ya que cada thread le entrega el turno al siguiente con la línea `turnoActual=(i+1) %3`. Con esto, es posible que un thread `i` le entregue el turno al thread `(i+1) %3` y antes que este último salga de la SC thread `(i-1) %3` le entregué el turno al thread `i`.
- Espera acotada **no se cumple**, un thread `i` puede ceder el turno al thread `(i + 1) % 3`, pero antes de que logre entrar a la SC, el thread `(i-1) %3` cede el turno al thread `i`, luego el `(i+1) %3` no entrará, esperarando infinitamente.
- Progreso **no se cumple**, si un thread `i` e `(i+1) %3` quieren entrar a la SC y el `turnoActual` es igual a `(i-1) %3`, y este thread se encuentra en `ContinuarTarea(i)`, entonces los otros dos se quedarán esperando.

Memoria y Archivos - Parte I

Glosario - Memoria

1. Memoria
   - Definición: Es un recurso del sistema que almacena datos y programas temporales, permitiendo al CPU acceder rápidamente a la información necesaria para la ejecución de tareas.
2. Direcciones de Memoria
   - Definición: Son números únicos que identifican ubicaciones en la memoria donde se pueden almacenar o recuperar datos.
3. Espacio de Direcciones
   - Definición: Es el rango completo de direcciones de memoria que un proceso puede utilizar. Depende de la arquitectura del sistema (por ejemplo, 32 bits o 64 bits).
4. Compactación
   - Definición: Es una técnica utilizada para reorganizar la memoria y eliminar los espacios vacíos, permitiendo que el sistema operativo asigne bloques contiguos a los procesos.
5. Fragmentación
   - Definición: Es la condición en la que la memoria se divide en bloques de diferentes tamaños, lo que puede resultar en un uso ineficiente de la memoria.
   - Tipos:
     - Fragmentación Interna: Ocurre cuando se reserva más memoria de la necesaria, creando espacio no utilizado dentro de un bloque asignado.
     - Fragmentación Externa: Ocurre cuando hay suficiente memoria total libre, pero no está contigua, lo que impide la asignación de un bloque grande.
6. Segmentación
   - Definición: Es un método de gestión de memoria que divide la memoria en segmentos lógicos, donde cada segmento representa una unidad de información (como una función o un array).
7. Implementación de Segmentación
   - Definición: Consiste en el uso de tablas de segmentos, donde cada entrada incluye la dirección base y el límite del segmento, facilitando la gestión y acceso a diferentes partes de la memoria.
8. Paginación
   - Definición: Es un método de gestión de memoria que divide el espacio de direcciones en páginas de tamaño fijo, permitiendo que las páginas se mapeen a la memoria física de manera no contigua.
9. Paginación con TLB (Translation Lookaside Buffer)
   - Definición: Es una técnica que utiliza una memoria caché rápida para almacenar las traducciones de direcciones virtuales a físicas, mejorando la velocidad de acceso a las páginas.
10. Efectividad de TLB
    - Definición: Es una medida del rendimiento de la TLB, generalmente expresada como la tasa de aciertos (hit rate), que indica el porcentaje de veces que la TLB tiene la traducción requerida.
11. Variantes de Paginación
    - Definición: Diferentes enfoques de paginación que pueden incluir diferentes tamaños de página, niveles de paginación y técnicas de optimización.
    - Tipos:
      - Fragmentación Interna: Ocurre cuando hay espacio no utilizado dentro de una página asignada.
      - Segmentos Páginados: Combina segmentación y paginación, donde cada segmento se divide en páginas.
      - Tablas de Páginas: Son estructuras de datos que mapean direcciones virtuales a direcciones físicas.
      - Multinivel: Es un enfoque que utiliza varias tablas de páginas para reducir el espacio de memoria requerido.
12. Reemplazo de Páginas
    - Definición: Es un proceso que ocurre cuando un sistema necesita cargar una nueva página en la memoria y no hay suficiente espacio, por lo que se debe reemplazar una página existente.
    - Métodos:
      - Reemplazo Óptimo: Algoritmo que reemplaza la página que no se usará durante el tiempo más largo en el futuro.
      - FIFO (First In, First Out): Reemplaza la página más antigua en la memoria.
      - Random: Selecciona aleatoriamente una página para ser reemplazada.
      - LRU (Least Recently Used): Reemplaza la página que no se ha utilizado durante el mayor tiempo.
13. Workload con Loop Secuencial
    - Definición: Se refiere a un patrón de uso de memoria donde los procesos acceden a las páginas de memoria de manera secuencial, lo que puede influir en la eficiencia de los algoritmos de reemplazo de páginas.
14. Frame: Unidad de almacenamiento en la memoria física con tamaño fijo, utilizado en sistemas de paginación para almacenar páginas de procesos

Glosario - Archivos

1. Dispositivos de I/O (Entrada/Salida)
   - Definición: Son componentes del sistema que permiten la comunicación entre el sistema informático y el mundo exterior. Pueden ser dispositivos de entrada (como teclados y ratones) o de salida (como monitores e impresoras).
2. Sistemas de Disco
   - Definición: Son dispositivos de almacenamiento que utilizan discos magnéticos o sólidos (SSD) para guardar datos. Ejemplos incluyen discos duros (HDD) y unidades de estado sólido (SSD).
3. Scheduling de Accesos
   - Definición: Es la técnica utilizada para gestionar y optimizar el acceso a los dispositivos de almacenamiento, organizando las solicitudes de I/O de manera que se minimice el tiempo de espera y se maximice la eficiencia.
4. Sistemas de Archivos
   - Definición: Es una forma de organizar y almacenar archivos en un dispositivo de almacenamiento, permitiendo la gestión de datos y la recuperación eficiente de archivos. Define cómo se almacenan y recuperan los archivos.
5. Directorios
   - Definición: Son estructuras dentro de un sistema de archivos que organizan y agrupan archivos. Permiten a los usuarios y programas localizar y acceder a archivos de manera jerárquica.
6. API de Sistemas de Archivos
   - Definición: Interfaces de programación de aplicaciones que permiten a los desarrolladores interactuar con el sistema de archivos. Proporcionan funciones para crear, leer, escribir y eliminar archivos y directorios.
7. Links
   - Definición: En el contexto de sistemas de archivos, los links son referencias a archivos que permiten acceder a ellos desde diferentes ubicaciones. Existen dos tipos principales:
     - Hard Links: Referencias directas al mismo inode en el sistema de archivos, permitiendo que múltiples nombres de archivo apunten a los mismos datos.
     - Soft Links (o Symlinks): Referencias que apuntan a la ubicación de un archivo en el sistema, permitiendo que se puedan mover sin afectar el link.
8. Implementación de Sistemas de Archivos
   - Definición: Se refiere al diseño y la estructura subyacente que permite al sistema operativo gestionar el almacenamiento de archivos y directorios, incluyendo cómo se representan y acceden a los datos en el dispositivo.
9. Asignación de Bloques
   - Definición: Es el método utilizado para gestionar el espacio de almacenamiento en un dispositivo. Define cómo se asignan bloques de almacenamiento para archivos y cómo se organiza la memoria.
   - Métodos de Asignación:
     - Asignación Contigua: Los bloques de un archivo se almacenan de forma contigua, lo que permite un acceso rápido pero puede resultar en fragmentación.
     - Asignación Enlazada: Cada bloque contiene un puntero al siguiente bloque, lo que permite flexibilidad en el tamaño de los archivos pero puede causar un acceso más lento debido a la necesidad de seguir punteros.
     - Asignación Indexada: Utiliza una tabla de índices para gestionar bloques de archivos, donde un bloque especial (el bloque de índice) contiene punteros a los bloques de datos, mejorando la eficiencia en el acceso.

Memoria y Archivos - Parte II

Ejercicio

Considere un sistema cuyo hardware está diseñado para soportar hasta 8GB de memoria física, y el sistema operativo configurado para manejar páginas de 4KB y una TLB con 10 entradas. Los procesos tienen un máximo de memoria virtual de 128 MB.

a) Diseñe un sistema de direccionamiento de memoria usando tabla de páginas de un nivel. Especifique, describiendo claramente su cálculo la cantidad de bits necesarios para la dirección virtual, offset, número de página (VPN), número de frame (PFN), tamaño en memoria de la tabla de páginas, incluya un 4 bit de metadata. Considere además que la arquitectura requiere que cada entrada en la tabla de páginas (PTE) debe ser de un tamaño que sea múltiplo de 1 Byte (8 bits), esto significa que puede necesitar agregar bits adicionales (padding o alineamiento).

Respuesta

Dirercción virtual: Tenemos $128 \cdot 2^{20}$ direcciones posibles en la memoria virtual, por lo tanto son necesarios 27 bits.

offset: Tenemos $4 \cdot 2^{10}$ direcciones posibles dentro de una página, por lo tanto son necesarios 12 bits.

VPN: Dentro de la memoria virtual tenemos $128 \cdot 2^{20}/(4 \cdot 2^{10})$ páginas, por lo tanto son necesarios 15 bits.

PFN: Dentro de la memoria física tenemos $8 \cdot 2^{30}/(4 \cdot 2^{10})$ frames, por lo tanto son necesario 21 bits.

Tabla de páginas: El tamaño de la tabla de páginas igual a el tamaño de cada entrada ($|\text{PTE}|$), por la cantidad de entradas, y esto último es equivalente a la cantidad de páginas dentro de la memoria virtual. A continuación se muestra el detalle cálculo:

$$
\begin{aligned}
|\text{Tabla de Páginas}| &= |\text{PTE}| \cdot \#\text{páginas} \\
&= \left\lceil \frac{|PFN| + |metadata|}{8} \right\rceil B \cdot 2^{15} \\
&= \left\lceil \frac{21 + 4}{8} \right\rceil \cdot 2^{15} B \\
&= 4 \cdot 2^{15} B \\
&= 128 KB
\end{aligned}
$$