SISTEMAS OPERATIVOS

Ingeniería de Computadores 23-24

Práctica Kernel_Simulator

ÍNDICE:

ÍNDICE: **2**
Introducción **3**
Arquitectura del Sistema **3**
Archivo memory.c: 3
Archivo scheduler.c: 3
Archivo program_loader.c: 3
Archivo system_clock.c: 4
Estructuras de datos: 4
Timer 4
PCB 4
Kernel Machine 5
Planificador (scheduler) **5**
Pequeña introducción de qué es un scheduler: 5
Implementación del Scheduler 5
Política de Planificación Utilizada 6
Funcionamiento 6
Gestor de memoria **7**

Introducción

La arquitectura del sistema se basa en una serie de hilos que cooperan para simular el comportamiento de un kernel. Los principales componentes de esta arquitectura son el *Clock*, el *Timer*, el *Scheduler* y el *Generador de Procesos*. A continuación se describen las estructuras de datos y funciones y funcionamiento del simulador.

Arquitectura del Sistema

El sistema simula un kernel muy sencillo, este está compuesto por varios componentes, cada uno con funciones específicas, que interactúan, los temporizadores, la planificación de tareas, la carga de programas y el mantenimiento del reloj del sistema.

Archivo memory.c:

Se encarga de la gestión de la memoria. Su principal responsabilidad es la asignación y liberación de memoria, así como el mantenimiento de la tabla de páginas o una estructura similar que permita la gestión eficiente de la memoria. Este componente es crucial para que los programas cargados puedan utilizar la memoria disponible de manera organizada y segura.

El '*timer.c*' se dedica a la gestión de los temporizadores del sistema. Configura y maneja los temporizadores necesarios para la operación del kernel, generando interrupciones de tiempo que permiten al planificador cambiar de contexto entre los diferentes procesos. Esta función es esencial para asegurar que el sistema opere de manera multitarea.

Archivo scheduler.c:

Es responsable de la planificación de tareas. Este, gestiona la cola de procesos, seleccionando el próximo proceso a ejecutar y manejando los cambios de contexto entre procesos. La interacción con '*timer.c'* es vital aquí, ya que las interrupciones generadas por los temporizadores permiten al planificador decidir cuándo debe ocurrir un cambio de contexto, asegurando así que todos los procesos reciban el tiempo de CPU necesario para su ejecución.

Archivo program_loader.c:

Se encarga de la carga de programas en la memoria. Utilizando las funciones de '*memory.c*' para la asignación de memoria. Además, se ocupa de la inicialización de las estructuras necesarias para la ejecución de los programas, preparando todo lo necesario para que los programas puedan ser gestionados por el planificador.

Archivo system_clock.c:

Mantiene el tiempo del sistema. Provee la hora actual y se asegura de que los temporizadores gestionados por *'timer.c'* estén correctamente sincronizados. Esta sincronización es crucial para el correcto funcionamiento de los temporizadores y, por ende, para la planificación de tareas.

El flujo de trabajo del sistema comienza con la inicialización, donde *'program_loader.c'* carga los programas en la memoria. Después se asigna la memoria necesaria para cada programa, que luego son gestionados por scheduler, organizándolos en la cola de procesos. Aquí el timer establece los temporizadores que permiten a scheduler realizar cambios de contexto periódicamente. Durante todo este proceso, el reloj mantiene el tiempo del sistema, asegurando la precisión temporal necesaria para el correcto funcionamiento de los temporizadores y del planificador.

Estructuras de datos:

A continuación, analizaremos la estructura de datos.

Timer

Esta estructura del timer incluye tres campos: '*target_pulse'*, que especifica el objetivo de pulsos para el temporizador; '*pulse_counter'*, que cuenta los pulsos actuales; y *routine*, que es un puntero a la función que se ejecutará cuando el temporizador alcance el objetivo.

```
// Estructura del temporizador
struct timer
{
   unsigned long target_pulse;
   unsigned long pulse_counter;
   void (*routine)();
};
```

PCB

La estructura pcb encapsula toda la información necesaria para controlar un proceso. Contiene un puntero next para formar listas enlazadas de PCBs, un identificador de proceso pid, el estado actual del proceso, un quantum, la dirección del contador de programa pc y una estructura MM para la gestión de memoria del proceso.

```
enum state {NEW, READY, RUNNING};
struct PCB {
   struct PCB *next;
```

```
    int pid;
    enum state state;
    int quantum_ms;
    address pc;
    int registers[REGISTERS_COUNT];
    struct MM mm;
};
```

Kernel Machine

Incluye la velocidad del reloj (clock_rate), la frecuencia del planificador (scheduler_rate), la tasa de generación de procesos (process_generator_rate), el número de CPUs (num_CPUs), núcleos por CPU (cores_per_CPU), hilos por núcleo (threads_per_core), y un puntero a las CPUs (CPUs).

```
struct kernel_machine {
    unsigned clock_rate;
    unsigned scheduler_rate;
    unsigned process_generator_rate;
    int num_CPUs;
    int cores_per_CPU;
    int threads_per_core;
    struct CPU *CPUs;
};
```

Planificador (scheduler)

Pequeña introducción de qué es un scheduler:

El scheduler es uno de los componentes más importantes en la simulación de este kernel, ya que se encarga de la planificación y gestión de las tareas o procesos que deben ejecutarse en el sistema.

A continuación, se detallará la implementación del planificador, la política de planificación utilizada y su funcionamiento.

Implementación del Scheduler

El planificador (scheduler) es responsable de gestionar la cola de procesos listos para ejecutar y de decidir cuál de ellos debe ser ejecutado en cada momento. Esto incluye la gestión de la estructura de datos que representa los procesos (típicamente una cola de procesos listos), el manejo de los cambios de contexto y la implementación de la política de planificación.

El scheduler utiliza una estructura de datos PCB (Process Control Block) que contiene información esencial sobre cada proceso, como su identificador de proceso (PID), su prioridad, su estado actual (listo, ejecutando, bloqueado), el contador de programa y los registros de la CPU.

Política de Planificación Utilizada

Existen varias políticas aplicables al planificador, en esta simulación se ha usado la combinación de Round-Robin (RR).

El Round-Robin es una política de planificación que asigna un tiempo fijo, cuanto de tiempo tiene cada proceso en la cola de procesos. Una vez que el proceso ha utilizado su quantum de tiempo, se suspende y se coloca al final de la cola de procesos, permitiendo que el siguiente proceso en la cola sea ejecutado.

Funcionamiento

El funcionamiento del planificador puede describirse en varias etapas:

1. **Inicialización**: Al inicio, el planificador configura la estructura de datos necesaria, como la cola de procesos listos y la tabla de PCBs. También inicializa los contadores y punteros que serán utilizados durante la planificación.
2. **Añadir Procesos a la Cola**: Cuando un nuevo proceso es cargado en el sistema (por ejemplo, mediante el módulo program_loader), se crea un nuevo PCB para ese proceso y se añade a la cola de procesos.
3. **Selección del Proceso a Ejecutar**: El planificador revisa la cola de procesos para encontrar el siguiente proceso a ejecutar.
4. **Cambio de Contexto**: Cuando se selecciona un nuevo proceso para ejecutar, el planificador realiza un cambio de contexto. Esto implica guardar el estado del proceso actual (sus registros, contador de programa, etc.) y cargar el estado del nuevo proceso seleccionado.
5. **Ejecución del Proceso**: El proceso seleccionado se ejecuta por un tiempo determinado. Si el proceso termina su ejecución antes de que expire su quantum, se elimina de la cola. Si el proceso necesita más tiempo, se suspende y se coloca al final de la cola de procesos.
6. **Manejo de Interrupciones**: El planificador también responde a las interrupciones generadas por los el timer. Estas interrupciones son esenciales para implementar el quantum de tiempo de la política Round-Robin. Cuando ocurre una interrupción, el planificador verifica si el quantum de tiempo del proceso actual ha expirado y, si es así, realiza un cambio de contexto.

Gestor de memoria

La memoria se reserva al inicio del simulador, dividiéndose en frames para su gestión. Los frames pueden ser asignados y liberados tanto en la memoria de usuario como en la memoria del kernel, utilizando funciones específicas para cada tipo de memoria.

Los hilos tienen un PTBR y una TLB que les permite realizar traducciones rápidas de direcciones virtuales a físicas. Si una página no se encuentra en la TLB, se busca en la tabla de páginas y, de ser necesario, se asigna un nuevo frame y se actualiza tanto la tabla de páginas como la TLB.

El loader lee los programas y asigna frames en las zonas de usuario y kernel, configurando las traducciones necesarias para las secciones de código y datos. El PCB ahora incluye una estructura MM que almacena las direcciones virtuales del código y datos, así como la dirección física de la tabla de páginas.

El reloj del sistema ejecuta una función que procesa las instrucciones de los hilos en ejecución, asegurando la correcta sincronización y ejecución de los procesos planificados. La planificación y ejecución de procesos se gestionan mediante funciones que seleccionan el próximo proceso a ejecutar e interrumpen el proceso actual si es necesario, manteniendo así el flujo del sistema.