Avances del Proyecto

Comandos en Linux

Para optimizar el trabajo durante el desarrollo, se han identificado diversos comandos de Linux que resultan esenciales para el análisis, la depuración y la optimización de programas. A continuación, se presentan algunas de las herramientas más relevantes, junto con sus respectivas funcionalidades y ejemplos de uso:

1. Medición de Tiempo

El comando **time** permite medir la duración de la ejecución de un programa, ofreciendo información detallada sobre el tiempo de CPU empleado, el tiempo real transcurrido y otros parámetros. Esta herramienta es fundamental para evaluar el rendimiento y detectar cuellos de botella en la ejecución.

Ejemplo de uso:

time ./RUN

2. Detección de Fugas de Memoria

Para identificar fugas de memoria y errores en la gestión de la misma, se utiliza **Valgrind** con la herramienta **memcheck**. Esta herramienta es la estándar para detectar errores relacionados con la memoria, proporcionando informes detallados sobre asignaciones incorrectas, liberaciones olvidadas y otros problemas de memoria.

Ejemplo de uso:

valgrind --leak-check=full ./RUN

3. Detección de Condiciones de Carrera

Helgrind está diseñado para detectar condiciones de carrera en programas multihilo. Ayuda a identificar accesos concurrentes conflictivos a variables compartidas, facilitando la corrección de problemas de sincronización.

Ejemplo de uso:

valgrind --tool=helgrind ./RUN

Idea Principal del Mostrador

El sistema se fundamenta en la lectura y distribución eficiente de equipajes a través de una arquitectura concurrente. En este contexto, la cola FIFO simple se genera a partir de la lectura de todos los equipajes ubicados en la carpeta de prueba del proyecto. Esta cola actúa como la fuente única de datos para el procesamiento posterior.

```c
void leer_entradas(const char *filename) {
    Equipaje equipaje;
    FILE *file = fopen(filename, "r");
    char buffer[256];
    construtorEquipaje(&equipaje);
    if (file == NULL) {
        perror("Error abriendo el archivo");
        exit(EXIT_FAILURE);
    }
    while (fgets(buffer, sizeof(buffer), file) != NULL) {
        construtorEquipaje(&equipaje);
        if (sscanf(buffer, "%s %s %s %f ", equipaje.tipo, equipaje.ciudad, equipaje.pais,&equipaje.peso)) {
            encolar(&pasajeros, equipaje);
        }
    }
    fclose(file);
}
```

Un total de 5000 mostradores se encargan de leer, de forma concurrente, los equipajes de dicha cola FIFO. Esta concurrencia en la lectura permite que el sistema procese un gran volumen de equipaje de manera ágil y simultánea, garantizando un flujo constante de datos hacia las siguientes etapas del proceso.

Considerando la existencia de 500 cintas transportadoras en el sistema, se ha implementado una estrategia de agrupación: cada conjunto de 10 mostradores se asocia a una única cinta transportadora. Esta cinta no solo transporta el equipaje, sino que además se encarga de su clasificación. Con este diseño, se logra que cada cinta opere de manera independiente, utilizando su índice correspondiente para acceder a su cola específica.

Para simular el proceso de entrega y clasificación del equipaje, se emplea un arreglo estático de colas, en el cual cada cola representa la línea de espera asignada a una cinta transportadora. Este enfoque asegura la concurrencia entre las cintas, optimizando el procesamiento y la organización del equipaje en el sistema.

```c
#define MAX_MOSTRADORES 5000
#define MAX_CINTAS 500
#define MAX_EQUIPAJES 120736

void *mostrador(void *args);
void leer_entradas(const char *filename);

sem_t semMostrador;
Cola pasajeros,cinta[MAX_CINTAS];
```

En resumen, la implementación propuesta garantiza que, a través de 5000 mostradores concurrentes, la lectura inicial de equipajes desde una cola FIFO (generada a partir de la carpeta de prueba) se distribuya de manera organizada hacia 500 cintas transportadoras, cada una de las cuales, mediante su respectiva cola, realiza el transporte y clasificación de los equipajes de forma autónoma y paralela.

Desarrollo del Mostrador

El desarrollo del mostrador inicia con la creación de 5000 hilos que se ejecutan concurrentemente, implementados mediante un bucle (for) que genera cada uno de estos hilos para simular la concurrencia en la lectura y distribución de equipajes. Esta estrategia es fundamental para representar el comportamiento real de los mostradores en el sistema.

```
Cola res;
pthread_t mostradores[MAX_MOSTRADORES];

printf("\n\t\tBienvenido\n");
sem_init(&semMostrador,1);

crear(&pasajeros);
leer_entradas("../Pruebas/text.txt");
printf("\t===1 leido===\n");

for (i = 0; i < MAX_MOSTRADORES; i++){
    pthread_create(&mostradores[i],NULL,mostrador,&i);
}

for (i = 0; i < MAX_MOSTRADORES; i++){
    pthread_join(mostradores[i],NULL);
}

sem_destroy(&semMostrador);
```

En un primer intento de sincronización, se utilizó el algoritmo de la panadería. Sin embargo, durante las pruebas se detectaron múltiples inconvenientes, tales como condiciones de carrera (race conditions) y problemas de inanición, lo cual comprometía la estabilidad y eficiencia del sistema.

Ante estos desafíos, se optó por cambiar la estrategia de sincronización hacia el uso de semáforos Binarios. Inicialmente se implementó una librería de semáforos predeterminada, pero esta presentó problemas de asignación: de las 500 cintas transportadoras disponibles, menos de la mitad se utilizaban adecuadamente, lo que evidenciaba una distribución subóptima de los recursos.

```
void *mostrador(void *args){
    int id = *((int *)args),i,indice = 0;
    while(1){
        sem_wait(&semMostrador);
        if(band){
            Cola equipaje;
            nroEquipaje++;
            pasajeros.primero->info.id = nroEquipaje;

            indice = id/10; //distribucion de cintas
            equipaje = cinta[indice];

            encolar(&equipaje,primero(pasajeros));
            cinta[indice] = equipaje;
            desencolar(&pasajeros);

            cantidad++;
        }
        if (esVacio(pasajeros)){
            band = 0;
            sem_post(&semMostrador);
            pthread_exit(NULL);
        }else{
            sem_post(&semMostrador);
        }
    }
}
```

Para solucionar esta limitación, se desarrolló e implementó una librería propia de semáforos. Con esta mejora, se logró una distribución más equitativa y eficiente de la sincronización entre los hilos, permitiendo que la asignación de tareas a las cintas transportadoras se realice de manera óptima. Esta solución garantizó que los 5000 hilos pudieran coordinarse eficazmente, maximizando el rendimiento y la utilización de las cintas en el procesamiento concurrente del equipaje.

```c
typedef struct {
    int valor;
    pthread_mutex_t mutex;
    pthread_cond_t cond;
}sem_t;

int sem_init(sem_t *sem,int valor){
    sem->valor = valor;
    pthread_mutex_init(&sem->mutex,NULL);
    pthread_cond_init(&sem->cond,NULL);
}
int sem_destroy(sem_t *sem){
    pthread_mutex_destroy(&sem->mutex);
    pthread_cond_destroy(&sem->cond);
    return (0);
}
int sem_wait(sem_t *sem){
    pthread_mutex_lock(&sem->mutex);
    while (sem->valor <= 0){
        pthread_cond_wait(&sem->cond,&sem->mutex);
    }
    sem->valor--;
    pthread_mutex_unlock(&sem->mutex);
    return (0);

}
int sem_post(sem_t *sem){
    pthread_mutex_lock(&sem->mutex);
    sem->valor++;
    pthread_cond_signal(&sem->cond);
    pthread_mutex_unlock(&sem->mutex);
}
```

Bug

Durante las pruebas se observó que la asignación de cintas transportadoras presentaba una notable inestabilidad. En determinadas ejecuciones se detectó que podían quedar hasta 4 cintas sin utilizar, mientras que en otras no se asignaban hasta 500 cintas, existiendo variaciones significativas entre esos valores. Es importante destacar que dicha variación se reduce conforme aumenta la cantidad de equipajes procesados; es decir, a mayor volumen de equipaje, menor es la diferencia en el número de cintas vacías.

Esta inestabilidad en la asignación constituye uno de los principales problemas a abordar, ya que afecta la concurrencia y la eficiencia en la interacción entre el mostrador y las cintas transportadoras. Optimizar este comportamiento será esencial para garantizar un equilibrio adecuado en la distribución de los equipajes y mejorar el rendimiento global del sistema.

Evidencia Fotográfica