Sincronización de Hilos

Diaz Hernandez Braulio
Islas Gasca Ian Jair
González Escamilla Felipe Yael

Introducción

Objetivo: Explorar la sincronización de hilos en la programación concurrente y paralela.

Temas:

- Definición de hilos.
- Control de ejecución de múltiples hilos.
- Importancia de la ordenación y previsibilidad en entornos multihilo.
- Desafíos al compartir recursos y datos entre hilos.
- Técnicas de sincronización como mutex, semáforos, monitores y barreras.

Desarrollo

¿Qué son los Hilos?
Unidades de ejecución independientes dentro de un proceso.
Ejecutan tareas simultáneamente (en sistemas multinúcleo) o intercaladamente (en sistemas de un solo núcleo).

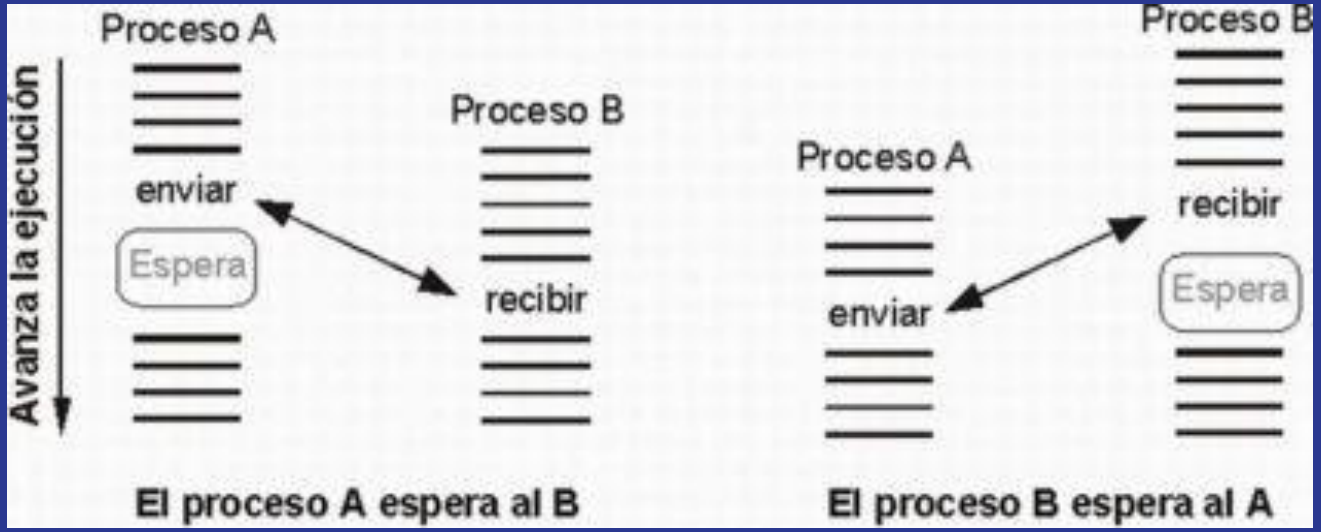
Problemas de la Programación Multihilo
Inconsistencia de datos.
Condiciones de carrera.
Bloqueos mutuos.

Técnicas de Sincronización de Hilos

- Mutex: Exclusión mutua.
- Semáforos: Control de acceso más fino.
- Monitores: Exclusión mutua y sincronización basada en condiciones.
- Barreras: Sincronización de un grupo de hilos en un punto específico.

Importancia de la Sincronización

- Garantiza la consistencia de datos.
- Evita problemas como deadlocks y race conditions.


Proceso A
Proceso B
Avanza la ejecución
enviar
Espera
recibir
El proceso A espera al B
Proceso A
Proceso B
recibir
enviar
Espera
El proceso B espera al A

Método Synchronized

Bloquea un objeto mientras se ejecuta un método sincronizado.
Evita que otros hilos accedan al objeto hasta que se desbloquee.

Ejemplo

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

struct myStruct{
int index;
char strinG[100];
int size;
 };

void *thread_routine(void *arg){ //Function to execute thread
struct myStruct* buffer = (struct myStruct *)arg; //Saved the structure of the casting char archivo_salida[] =
"salida.txt";

printf("%s\n", buffer->strinG);
//Create two pointers, the first to point the beginning and the second to point the final of the string char* initial = buffer->strinG;
char* final = buffer->strinG; int
sz = 0;
//Move the final pointer at the end of the string
```

```
while(*final != '\0') {
final++;
sz++;
  }
buffer->size = sz;
final--;

//Exchange characters from the ends to the center
while (initial < final) {
char temp = *initial;
*initial = *final;
*final = temp; initial++;
final--;
  }
printf("Thread: %d\n", buffer->index); printf("String:
%s\n", buffer->stringG); printf("Length: %d\n", buffer-
>size);

FILE* archivo_out = fopen(archivo_salida, "a"); if
(archivo_out == NULL) {
printf("No se pudo abrir el archivo de entrada.\n");
  }
fprintf(archivo_out, "Thread: %d\n", buffer->index);
fprintf(archivo_out, "String: %s\n", buffer->stringG);
fprintf(archivo_out, "Length: %d\n", buffer->size);
printf("Archivo creado correctamente.\n"); fclose(archivo_out);
```

```
printf(archivo_out, "Thread: %d\n", buffer->index);
fprintf(archivo_out, "String: %s\n", buffer->strinG);
fprintf(archivo_out, "Length: %d\n", buffer->size);
printf("Archivo creado correctamente.\n"); fclose(archivo_out);
pthread_exit(NULL);
 }

int main(){
int numberThreads;
char archivo_entrada[] = "entrada.txt"; printf("How
many threads wanna create? \n"); scanf("%d",
&numberThreads);


struct myStruct Data[numberThreads]; //Create a data structure pthread_t  idThread[numberThreads];
//Create a thread

for(int i = 0; i < numberThreads; i++){
Data[i].index = i;

printf("Enter a string for the thread %d: ", Data[i].index); scanf("
%[^\n]", Data[i].strinG);
```

```
* Function to create thread 1st
argument: Pointer to the id
2nd argument: Thread attributes
3rd argument: Routine that is executed when the thread is created
4th argument: Thread function input arguments (Here we casting the structure "(void *)")*/ if(0 !=
pthread_create(&idThread[i], NULL, thread_routine, (void *)&Data[i]))
printf("\n Error");
FILE* archivo_in = fopen(archivo_entrada, "a"); if  (archivo_in == NULL) {
printf("No se pudo abrir el archivo de entrada.\n"); }
fprintf(archivo_in, "%s\n", Data[i].strinG);
printf("Archivo creado correctamente.\n");

fclose(archivo_in);
  }

for(int i = 0; i < numberThreads; i++){ pthread_join(idThread[i], NULL); //Function to
wait father
  }

return 1;
}
```

Descripción del ejemplo

Creación de hilos:

El código permite crear múltiples hilos según la cantidad especificada por el usuario. Cada hilo ejecuta la función thread_routine, que realiza operaciones en una estructura compartida.
Manipulación de cadenas de caracteres: La función thread_routine invierte la cadena de caracteres ingresada por el usuario y guarda el resultado en la estructura myStruct. Además, se registra la longitud de la cadena invertida.
Lectura de entrada y escritura de salida: El código permite al usuario ingresar una cadena de caracteres para cada hilo y la guarda en un archivo de entrada. También guarda la cadena invertida y su longitud en un archivo de salida.

Conclusión

- La sincronización de hilos es esencial para la programación concurrente.
- Diversas técnicas y algoritmos abordan la sincronización.
- Elección de la técnica adecuada según requisitos y complejidad del problema.