Logo

Description automatically generated

Axel Roberto Orozco Hernández

NAVARRO QUINN, LUIS ROBERTO

Fundamentos de Sistemas Operativos

Profesor: Leonardo Sandoval. Gonzalez

Actividad 17

**1.-** Del problema de los 4 jugadores de dominó sentados en una mesa jugando dominó, se van turnando para tirar una ficha o pasar si no pueden jugar en el sentido contrario a las manecillas del reloj. Ahora mediante pseudocódigo represente a los cuatro jugadores con procesos desde P(0) hasta P(3) y como sincronizaría los turnos usando un sistema de paso de mensajes.

Inicializar cola\_mensaje; **//Crea una cola de mensajes para la comunicación.**

Proceso P(i)

Mientras (partida\_no\_ha\_terminado)

Recibir mensaje de turno; **//Esperar el mensaje que indica que es su turno.**

Si(puede\_jugar)

Jugar ficha;

Sino

Pasar turno;

Enviar mensaje a P((i-1) % 4); **//Pasar el turno al siguiente jugador en orden inverso.**

Fin Mientras

Fin Proceso

**//Inicializar el juego**

Enviar mensaje inicial a P(0); **//Empiza el jugador P(0)**

**2.-** Codifica el algoritmo usando el sistema de paso de mensajes de la librería Posix

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#include <signal.h>

#define NUM\_JUGADORES 4

#define MSG\_KEY 1234

typedef struct {

long tipo;

int jugador;

} mensaje\_t;

int id\_cola\_mensajes;

//Funcion para simular un tiempo de espera

void jugar(int jugador) {

printf("Juagador %d esta jugando su turno\n", jugador);

unsleep(rand() % 1000000); // Simula un tiempo de espera aleatorio

}

//Funcion que ejecuta cada jugador en su proceso

void jugador\_proceso(int jugador){

mensaje\_t mensaje;

int siguiente\_jugador = (jugador - 1 + NUM\_JUGADORES) % NUM\_JUGADORES;

while (1) {

msgrcv(id\_cola\_mensajes, &mensaje, sizeof(mensaje.jugador), jugador + 1, 0);

if(rand() % 2 == 0){

jugar( jugador );

}else{

printf("Jugador %d pasa el turno\n", jugador);

}

//Pasar el turno al siguiente jugador

mensaje.tipo = siguiente\_jugador + 1;

mensaje.jugador = siguiente\_jugador;

msgsnd(id\_cola\_mensajes, &mensaje, sizeof(mensaje.jugador), 0);

}

}

//Funcion que ejecuta el proceso padre

void limpiar(int signo){

msgctl(id\_cola\_mensajes, IPC\_RMID, NULL);

exit(0);

}

int main(){

int i;

mensaje\_t mensaje;

//Creacion de la cola de mensajes

id\_cola\_mensajes = msgget(MSG\_KEY, 0600 | IPC\_CREAT);

if(id\_cola\_mensajes == -1){

perror("Error al crear la cola de mensajes");

exit(1);

}

//Manejar la señal de interrupcion para limpiar la cola de mensajes

signal(SIGINT, limpiar);

//Crear los procesos hijos

for(i = 0; i < NUM\_JUGADORES; i++){

if(fork() == 0){

jugador\_proceso(i);

exit(0);

}

}

//Inicializar el juego enviando el primer mensaje al primer jugador

mensaje.tipo = 1;

mensaje.jugador = 0;

msgsnd(id\_cola\_mensajes, &mensaje, sizeof(mensaje.jugador), 0);

//Esperar a que terminen los procesos hijos

for(i = 0; i < NUM\_JUGADORES; i++){

wait(NULL);

}

//Limpiar la cola de mensajes

limpiar(0);

return 0;

}

**3.-** ¿Qué aprendiste?

Manejo de Paso de mensajes con POSIX: Aprendimos como usar las colas de mensajes POSIX para comunicar procesos entre si.

Sincronizacion y exclusion mutua: Aprendimos como sincronizar procesos para que se ejecuten en un orden determinado y como evitar condiciones de carrera.

Modelado de problemas concurrentes: Aprendimos como modelar problemas concurrentes y como implementar soluciones para estos problemas.