**Centro Universitario de Ciencias Exactas e Ingenerias**

**Departamento de Ciencias Computacionales**

**Seminario de Sistemas Operativos**



**Violeta del Rocio Becerra Velazquez**

**Edgar Agustin Martinez Gonzalez**

**220286695**

**Ingeneria en Computacion**

**D01**

**Simular el procesamiento por lotes con Multiprogramación.**

**11/09/2022**

En la practica anterior creamos una clase State para organizar la informacion que se vera en la aplicacion y mandar a ejecutar cada paso del programa. En esta clase es donde mandaremos a llamar las interrupciones y los erroresque el usuario invoque desde el teclado.

Cuando se invoca la interrupcion la clase State llama al metodo interrupt del Batch actual, el batch intercambiara de posicion el proceso actual y el siguiente.

pub fn interrupt(&mut self) {

if self.current\_process < 2 {

let temp = self.processes[self.current\_process].clone();

self.processes[self.current\_process] = self.processes[self.current\_process + 1].clone();

self.processes[self.current\_process + 1] = temp;

}

}

Cuando se invoca un error la clase State llama al metodo end\_without\_result del Process actual que anulara el resultado y el tiempo restante del proceso (la interfaz grafica imprime como “Error” los resultados nulos).

pub fn end\_without\_result(&mut self) {

self.remaining\_time = 0;

self.operation.result = *None*;

}

Para poder invocar estas funciones se ha agregado la lectura del teclado durante el ciclo de pasos de 1 segundo. Como se puede ver a continuacion la interrupcion (‘e’) y el error (‘w’) se mandan a llamar desde el estado (state).

if poll(Duration::*from\_secs*(1))? {

if let Event::*Key*(key) = read()?{

match key.code {

KeyCode::*Char*('q') => return *Ok*(()),

KeyCode::*Char*('p') => pause = true,

KeyCode::*Char*('c') => pause = false,

KeyCode::*Char*('w') => state.end\_process\_without\_result(),

KeyCode::*Char*('e') => state.interrupt(),

\_ => {}

}

}

}

Para la pausa (“p”) y continuar (“c”) lo unico que se hace es cambiar una variable booleana que indica en el ciclo si se debera ejecutar un siguiente paso o no dependiendo de si quedan lotes o el programa esta en pausa

if !state.is\_finished() && !pause {

state.execute\_step();

}

Finalmente para crear los procesos aleatoriamente hemos creado un constructor de State que crea procesos con valores aleatorios obtenidos con la libreria Rand.

pub fn *new*(num\_processes: usize) -> State {

let mut processes = Vec::*new*();

let mut rng = rand::thread\_rng();

for i in 0..num\_processes {

processes.push(

Process::*new*(

i as u8,

rng.gen\_range(5..15),

Operation::*new*(

rng.gen\_range(0..15) as usize,

rng.gen\_range(1..15) as usize,

rng.gen::<Operator>()

)

)

);

}

State {

batches: Batch::*batches\_from\_processes*(processes),

current\_batch: 0,

time\_on\_execution: 0,

}

}