## Proceso

prioridad : Entero

tiempoEjecucion : Entero

tiempoEjecutado : Entero

periodoInterrupcionES : Entero

largoInterrupcionES : Entero

tiempoEnES : Entero

UserOrSO : Booleano

id : String

nombre : String

estados : static enum

estadoActual : estados

updateEstadoActual:

El método analiza el estado del proceso y determina su estado y actualiza de ser necesario. Devuelve el estado nuevo o el mismo que tenia antes, si no hay cambios.

PreCondiciones:

Al proceso se le asignaron sus atributos de manera correcta y no nula

El proceso cuenta con un enumerador estático que representa los diferentes estados posibles

PostCondiciones:

El proceso actualiza su estado, de ser necesario

El proceso devuelve su estado actual

Pseudo:

Proceso.updateEstadoActual() : devuelve Proceso.Estado

COM

Si estadoActual = Bloqueado

Devuelve estadoActual

FINSI

SI tiempoEjecutado >= tiempoEjecucion

This.estado <- Finalizado

Devuelve finalizado

SINO SI periodoInterrupcionES > 0 y tiempoEjecutado mod periodoInterrupcionES = 0 y estadoActual = Listo

This.estado <- BloqueadoES

Devuelve BloqueadoES

SINO SI tiempoEnES >= largoInterrupcionES y estadoActual = BloqueadoES

estadoActual <- Listo

tiempoEnES <- 0

devuelve Listo

SINO

Devuelve estadoActual

FINSI

FIN

### getData

Devuelve un string formateado que da toda la información de estado del proceso

Precondiciones:

El proceso tiene todos sus datos asignados correctamente y no nulos

PostCondiciones:

El proceso no se modifica

Devuelve un String con toda su información

Pseudo:

Proceso.getData() : Devuelve String

COM

Res <- “ ”

Res <- res + “Nombre: ” + getNombre()

Res <- res + “ID: “ + getID()

Res <- res + “Estado: “ + getEstadoActual()

Res <- res + “Nivel de Prioridad: “ + getPrioridad()

Res <- res + “Tiempo Ejecución: “ + getTiempoEjecutado() + “/” + getTiempoEjecucion()

Res <- res + “Periodo de Interrupción: “ + getPeriodoES()

Res <- res + “Largo de Interrupción: “ + getLargoES()

Si (UserOrSO = SO)

Res <- res + “Es de nivel: SO”

SINO

Res <- res + “Es de nivel USER”

FINSI

Devuelve Res

FIN

## Planificador

listaListos : LinkedList<Proceso>[99]

Bloqueados : AdminProcBloqueados

Logger : String

numRonda : Entero

quantum : Entero

procesadoresExistentes : LinkedList<CPU>

despachar(Proceso proc) : void

revisarBloqueados() : void

### pasarAEjecutables()

Recorre los CPUs asociados al planificador y verifica si se les puede cargar un proceso nuevo, si se puede, se le cargan los siguientes procesos en cola. Cuando se carga un proceso se añade al logger

Precondiciones:

El planificador tiene CPUs y procesos cargados

PostCondicones

Se cargan procesos nuevos en los CPUs disponibles

Pseudo:

Planificador.pasarAEjecutables()

COM

cpuDisponibles : Lista

PARA CADA cpu en procesadoresExistentes

Si cpu.pideCambio o cpu.getProcesoCargado = nulo

cpuDisponibles.añadir(cpu)

FINSI

FINPARACADA

SI cpusDisponibles.esVacia

Termina

FINSI

PARA i = 0, i hasta 99

PARA CADA proceso en listaListos[i]

cpusDisponibles[0].setProceso(proceso)

cpusDisponibles.sacar(cpusDisponibles[0])

logger.añadir(“Pase ” + proceso + “a CPU”)

FINPARACADA

FINPARA

FIN

### pasarBloqueadosAListos

toma los procesos del administrador de procesos bloqueados que se hayan desbloqueado durante el ultimo quantum y los coloca en la cola de listos.

Precondiciones:

Un administrador de procesos bloqueados

Una cola de multiple nivel de listos

Postcondiciones:

La cola de procesos listos crece con los procesos que se desbloquearon o queda igual

Pseudo:

Planificador.pasarBloqueadosAListos():devuelve Lista

COM

Desbloqueados <- AdminProcBloqueados.getDesbloqueados(quantum)

i <- 0

Para cada cola en desbloqueados

listaListos[i].añadirTodos(cola)

i++

FINPARACADA

Devuelve desbloqueados

FIN

## CPU

procesoCargado : Proceso

pideCambio : Booleano

ID : entero

### ejecutarProceso:

Cuando el planificador le despacha al CPU un proceso, este simula ejecutarlo sumando al atributo de tiempoEjecutado del mismo, de manera secuencial y verificando si en algún momento se bloquea el mismo por si solo o si finaliza. Una vez que ejecuto todos los ciclos que le corresponden, devuelve el estado final en el que quedo el proceso.

Precondiciones:

Hay un quantum definido

EL CPU tiene un Proceso Cargado

El proceso esta listo para ejecución

Postcondiciones:

El proceso tiene “quantum” mas en su tiempo ejecutado

A lo largo de la ejecución, el proceso puede bloquearse o finalizar, por lo que su ejecución termina

Pseudo:

CPU.ejecutarProceso(Entero quantum)

COM

SI procesoCargado <> nulo

Para cada i desde 0 hasta quantum

Proceso.tiempoCargado++

Estado <- Proceso.updateEstadoActual()

Si Estado <> Listo

procesoCargado <- nulo

pideCambio <- true

termina

FINSI

FIN PARACADA

pideCambio <- true

FIN

## AdminProcBloqueados

procesosBloqueados : LinkedList<Proceso>

### getDesbloqueados:

Una vez un proceso haya pasado por el tiempo suficiente para desbloquearse, el proceso debe de pasar a la cola de listos. Para hacer esto, cada vez que se ejecutan los procesos en CPU, la misma cantidad de tiempo que se ejecutan los procesos en CPU debe de pasarse en la cantidad de tiempo que los procesos pasan bloqueados, para mantener una línea temporal.

Precondiciones:

Hay un quantum definido

Hay elementos en la cola de bloqueados

Postcondiciones

El proceso tiene el quantum agregado al tiempo que pasó bloqueado

Si el tiempo que pasó bloqueado equivale al tiempo necesario para estar bloqueado, debe de desbloquearse.

Pseudo:

AdminProcBloqueados.getDesbloqueados(Entero quantum) : LinkedList[]

COM

Resultado <- nueva LinkedList[99]

Contador <- 0

Mientras contador <99 hacer

Resultado[Contador] <- nueva LinkedList()

Contador <- Contador + 1

FIN Para Cada

Contador <- 0

Aux <- nueva LinkedList<Proceso>()

Mientras Contador < Quantum hacer

Para cada Proceso en procesosBloqueados hacer

Si Proceso.Estado = Bloqueado entonces

Proceso.agregarTiempoEnES(1)

Si NOT Aux.Contiene(Proceso) AND Proceso.updateEstadoActual <> Bloqueado hacer

Resultado[Proceso.Prioridad-1].agregar(Proceso)

Aux.agregar(Proceso)

FinSi

SiNo

Si Proceso.Estado <> Bloqueado

Resultado[Proceso.Prioridad-1].agregar(Proceso)

Aux.agregar(Proceso)

FinSi

FinSi

Fin Para Cada

Contador <- Contador + 1

Fin Mientras

Para cada Proceso en Aux hacer

procesosBloqueados.eliminar(Proceso)

Fin Para Cada

Devolver Resultado

FIN

### printListaConMotivo:

Otorga un String con los elementos contenidos dentro de la lista de procesos bloqueados y la razón por la que están bloqueados.

Precondiciones

La lista de procesos bloqueados existe

Postcondiciones

No se ve alterada la lista de procesos bloqueados

Pseudo:

AdminProcBloqueados.printListaConMotivo() : String

COM

Resultado <- “”

Para Cada Proceso en procesosBloqueados hacer

Resultado <- Resultado + Proceso.Nombre + Proceso.ID + “:” + Proceso.Estado

Fin Para Cada

Si Resultado.Largo() > 3 entonces

Devolver resultado.Substring(0, Resultado.Largo()-2)

Fin Si

Devolver “”

FIN

### getProc:

Otorgar un proceso contenido dentro de la lista de procesos bloqueados basándose en la ID otorgada

Precondiciones

Existe una lista de procesos bloqueados

La ID otorgada es válida

Postcondiciones

Si encuentra la ID, otorga un proceso, sino, no otorga nada

La lista de procesos no debe verse afectada

Pseudo:

AdminProcBloqueados.getProc(String ID) : Proceso

COM

Para Cada Proceso en procesosBloqueados hacer

Si Proceso.ID = ID entonces

Devolver Proceso

Fin Si

Fin Para Cada

Devolver nulo

FIN

### listaBloqUser:

Obtener un vector de String de los elementos bloqueados

Precondiciones

Existe una lista de procesos bloqueadso

Postcondiciones

La lista de procesos bloqueados no debe de verse afectada

Pseudo:

AdminProcBloqueados.listaBloqUser() : String[]

COM

Resultado <- nuevo ArrayList<String>()

Para Cada Proceso en procesosBloqueados hacer

Si Proceso.Estado = Bloqueado entonces

Resultado.agregar(Proceso.Nombre + “:” + Proceso.ID)

FinSi

Fin Para Cada

Devolver Resultado.ConvertirA(nuevo String[0])

FIN