Certamen II
Lenguajes de Programación II – ICI425/INC415

Profesor: Alonso Inostrosa Psijas
Fecha: Jueves 12 de Octubre de 2023

Aspectos importantes:

- Programar en Golang, si desea utilizar otro lenguaje que admita concurrencia debe ser autorizado por el profesor.
- Desarrollar el control en grupos de máximo 3 integrantes. Los grupos pueden estar conformados por integrantes diferentes a evaluaciones anteriores.
- Debe crear un repositorio Git para respaldar su código, debe incluir un enlace al repositorio. No se admite como excusa el haber perdido el código desarrollado.
- Documente adecuadamente su programa usando comentarios en el código.
- Fecha de Entrega sugerida: 9 de Noviembre de 2023.
- Para más detalles, referirse al modelo de procesos de 5 estados en los libros Sistemas Operativos de los autores Andrew S. Tanenbaum o del William Stallings (ambos, de la editorial Pearson Educación).

Introducción

En un Sistema Operativo (SO), un proceso corresponde a una instancia de un programa en ejecución. Se caracteriza por una secuencia de instrucciones, un estado actual y recursos del sistema que se asocian a él. En esencia, un proceso consiste en el código del programa y un conjunto de datos. Por otra parte, es responsabilidad del SO asegurar que: Los recursos estén disponibles para múltiples procesos, conmutar la CPU para permitir la ejecución de varios procesos (de manera intercalada), y permitir que la CPU y los dispositivos de entrada-salida se utilicen de forma eficiente.

Una forma de hacerlo es combinando un modelo de procesos junto a una estructura de datos (asociada a cada proceso cargado) denominada “Bloque de Control de Programa” que contiene información que una entidad - llamada Dispatcher - del SO utiliza para controlar la transición del proceso por los diversos estados (ver Figura 1).

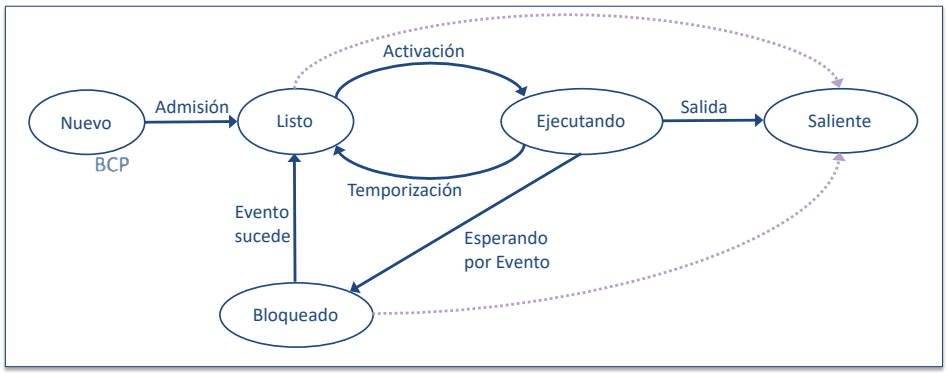

Figura 1- Diagrama de Transición de Estados del Modelo de Procesos de 5 Estados.

El *Dispatcher* otorga turnos de ejecución a cada proceso, a través de la conmutación del uso de la CPU. Para ello, realiza cambios de estados a cada proceso, de acuerdo con algún modelo de procesos, en nuestro caso, corresponde al de 5 estados (cuyas transiciones y estados se observan en el diagrama de la Figura 1).

Durante los instantes en que un proceso no está siendo ejecutado (es decir, no está en el estado "ejecutando"), estará almacenado en alguna de las dos colas correspondientes a aquellos que están en el estado "bloqueado" (cola de bloqueados) o en el estado "listo" (cola de listos), como se ilustra en la Figura 2. El *Dispatcher* selecciona el siguiente proceso para pasar al estado "ejecutando" desde la "Cola de Listos". Cuando un proceso en estado "Ejecutando" cumple con la ejecución de sus instrucciones correspondientes a su turno, luego pasa al estado "Listo" y es agregado a la "cola de listos". Cuando un proceso en estado "Ejecutando" encuentra una instrucción de E/S, es bloqueado y se lo almacena en "Cola de Bloqueados" hasta que el evento correspondiente a E/S se cumple, momento en el que puede pasar a "Cola de Listos" para ser – eventualmente - ejecutado.

Un proceso existe hasta que su última instrucción es ejecutada, momento a partir del cual debe pasar al estado "Saliente" y es eliminado (liberándose los recursos que le fueron asignados).

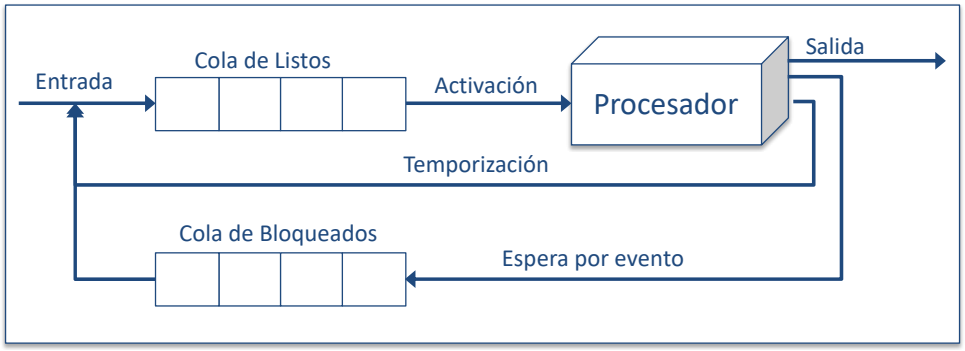

Figura 1- Esquema de 2 Colas para la Gestión de Procesos en el Modelo de Procesos de 5 Estados.

Problema 1: "Ejecución de m Procesos sobre Procesador con $n = 1$ núcleos"

Considerando un enfoque de Simulación Orientada a Procesos, y mediante el uso de corrutinas (implementadas mediante "gorrutinas" de Golang), desarrolle un programa que simule la ejecución de un modelo de procesos de 5 estados (ver Figura 1) siendo ejecutado sobre un sistema monoprocesador. Para ello, considere un esquema de 2 colas (ver Figura 2) para almacenar los procesos en sus respectivos estados, cada proceso deberá tener su propio BCP (sólo: Identificador de Proceso, Estado, Contador de Programa, Información de Estado E/S).

Existirá un Dispatcher que realizará el cambio de procesos cada m instrucciones de cada proceso. Para ello, dispone de un conjunto de instrucciones propias que le permiten realizar su función. Mediante ellas, el despachador deberá:

i) Guardar el estado del proceso (en el BCP) que estaba en ejecución (ST nombre_proceso).
ii) Enviar el proceso a la cola correspondiente según su estado (Listo: PUSH Listo – Bloqueado: PUSH Bloqueado).
iii) Deberá sacar el siguiente proceso a ejecutar desde la cola correspondiente PULL

iv) Cargará el proceso (LOAD nombre_proceso) y su estado (según el BCP) en la CPU.
v) Y finalmente dará la orden de inicio de ejecución del proceso (EXEC nombre_proceso).

Por lo tanto, un proceso consiste en sus instrucciones y su BCP. Dado que se dispone sólo de 1 núcleo ($n = 1$) simulado, un máximo de 1 proceso podrá estar en el estado "Ejecutando" en un instante cualquiera.

Entradas:

- *o* corresponde a la cantidad de instrucciones máximas a ejecutar por la CPU antes de cambiar a otro proceso (en caso de que existan más procesos en estado "Listo").
- Probabilidad P de que una instrucción del proceso en ejecución finalice antes de tiempo (término del proceso por motivos diferentes a la finalización normal). No corresponde a ninguna distribución estadística, sino simplemente como 1/P.
- Orden de Creación de los Procesos:
    - Corresponde a un archivo de entrada deberá contener los procesos y el orden temporal en que deberán ser creados.
    - Cada línea del archivo contendrá el tiempo (paso de CPU) en que el/los procesos deberán ser creados y el nombre del archivo que contiene las instrucciones del proceso (ver sección "Definición de Procesos"). Estos procesos deberán coincidir con el nombre de los procesos descritos en cada archivo de procesos (primera línea de cada archivo).

Salida:

La salida del programa será la traza de ejecución entrelazada de los procesos ejecutados. El programa deberá escribir en un archivo de texto el nombre del proceso y el nro. de línea de la instrucción ejecutada y la propia instrucción.

Definición de Procesos:

Cada proceso corresponde a un archivo de texto cuyo contenido es el siguiente:

- Nombre del programa debe ir en la primera línea del archivo.
- En líneas siguientes van las instrucciones. Dos tipos de instrucciones: i) instrucciones comunes, ii) instrucciones de llamada a Sistema (bloqueantes por espera de eventos o de E/S).
- Última instrucción del proceso debe ser de Finalización, para indicar al SO que termina su ejecución.

Ejecución:

El programa debe ser ejecutado en terminal mediante línea de comandos de la siguiente forma:

$ Orden_Ejecucion_Prog *n o P* archivo_orden_creacion_procesos nombre_archivo_salida

Formato Archivos:
Para cualquier archivo, si una línea comienza con #, entonces corresponde a comentarios y no debe ser considerada.

Tanto para las entradas como para las salidas puede utilizar la cantidad de archivos que considere apropiadas según el diseño de su simulador. Aún así, al menos debe considerar los siguientes archivos:

- **Orden de Creación de Procesos:** Contiene una marca de tiempo (o de instrucciones ejecutadas en total por la CPU) que indican el momento en que, uno o varios procesos, deben ser creados y añadidos a la gestión del despachador.
- **Procesos:** Corresponde un archivo que contiene las instrucciones que cada proceso debe realizar. Se definen de 1 instrucción por cada línea, y las instrucciones pueden ser del tipo I (instrucción normal, no causa interrupción), del tipo ES (instrucción de E/S, que bloquea al proceso por una cantidad c de instrucciones ejecutadas por la CPU.
- **Salida:** Corresponde a una traza de la ejecución de instrucciones por la CPU.

Contenido sugerido para archivo de orden de creación de procesos:

```
# Tiempo Creación (se agregan a Cola Listo)        Nombre de archivo del Proceso(s)
1       Proceso_1 Proceso_2
67      Proceso_3 Proceso_4 Proceso_5
120     Proceso_6 Proceso_7
166     Proceso_8 Proceso_9
```

Contenido sugerido para archivos de procesos:

```
# Tipo de Instrucción:
#       I       = Instrucción Normal.
#       ES c    = Instrucción E/S que se desbloqueará dentro n pasos de CPU (instrucciones
#               ejecutadas). Ocurrirá cuando haya cambio de proceso en la CPU, antes de cargar un
#               nuevo proceso.
#       F       = Finalizar. Dispatcher debe pasar el proceso al estado Saliente y eliminarlo.
#
# Contenido de Ejemplo:
# Nro Instrucción       Tipo de Instrucción
1       I
2       I
3       ES 7
4       I
5       I
6       F
```

Archivo de Salida (suponiendo $o = 4$):

```
# Tiempo de CPU        Tipo Instrucción Proceso/Despachador          Valor CP
1       PULL    Dispatcher      102
2       LOAD nombre_proceso_1           Dispatcher      103
3       EXEC    nombre_proceso_1        Dispatcher      104
4       I       nombre_proceso_1        8000
5       I       nombre_proceso_1        8001
6       I       nombre_proceso_1        8002
7       I       nombre_proceso_1        8003
8       ST nombre_proceso_1     Dispatcher      100
9       PUSH_Listo nombre_proceso_1     Dispatcher      101
10      PULL    Dispatcher      102
11      LOAD nombre_proceso_2           Dispatcher      103
12      EXEC    nombre_proceso_2        Dispatcher      104
13      I       nombre_proceso_2        6501
14      ES 7    nombre_proceso_2        6502
15      ST nombre_proceso_2     Dispatcher      100
16      PUSH_Bloqueado nombre_proceso_2     Dispatcher      101
17      PULL    Dispatcher      102
18      LOAD nombre_proceso_1           Dispatcher      103
19      EXEC    nombre_proceso_1        Dispatcher      104
20      I       nombre_proceso_1        8004
21      I       nombre_proceso_1        8005
22      I       nombre_proceso_1        8006
23      I       nombre_proceso_1        8007
24      ST nombre_proceso_1     Dispatcher      100
25      PUSH_Listo nombre_proceso_1     Dispatcher      101
26      ** EVENTO E/S – Mover Proceso desde Cola Bloqueado (nombre_proc) a Cola Listo **
27      PULL    Dispatcher      102
28      LOAD nombre_proceso_2           Dispatcher      103
29      EXEC    nombre_proceso_2        Dispatcher      104
30      I       nombre_proceso_2        6503
31      I       nombre_proceso_2        6504
32      I       nombre_proceso_2        6505
…
```

Problema 2: "Ejecución de m Procesos sobre Procesador con $n = x$ núcleos"

El funcionamiento es equivalente a Problema 1, pero la ejecución de los procesos se debe realizar considerando x núcleos, los que permiten la ejecución paralela de x procesos (es decir, un máximo de x procesos puede estar en el estado "Ejecutando" un determinado instante). Considere a x como un valor indicado por el usuario en la línea de comandos al momento de ejecutar el programa. Para determinar qué núcleo debe activar y ejecutar cada proceso, simplemente considere una estrategia

FCFS ("*First come – first served*"), es decir, el primero núcleo que se libera es el primero que toma el siguiente proceso de la cola de listos.
En la salida de este programa deberá, además de lo indicado en Problema 1, el número del núcleo que realizó la ejecución del proceso.

Observaciones:

- El lenguaje de programación debe ser Golang. Para otros lenguajes, debe consultar y ser aprobado por el profesor.
- Cada proceso simulado corresponde a una corrutina, al igual que el dispatcher.
- El programa termina cuando no quedan más procesos por ejecutar.
- Cuando un proceso padre finaliza, también deberán finalizar sus procesos hijos.
- Cada proceso se almacena en una dirección de memoria (cualquiera) y sus instrucciones se almacenan de forma secuencial. Ej: Si proceso se aloja en dirección 8000, su primera instrucción está en la dirección 8000, la siguiente en 8001, y así sucesivamente.
- El proceso despachador se almacena en la dirección 100.
- Se debe mostrar en el archivo de salida el punto en que ocurre un evento E/S, indicando el proceso que sale de la cola Bloqueado y pasa a cola Listo.
- De manera similar, se deben mostrar los procesos nuevos que se agregan a la cola Listo.

Todos los integrantes del equipo deben entender y poder explicar en su totalidad el código, por lo que deben repartirse el trabajo y comprender tanto la teoría como la práctica.
La copia entre equipos de trabajo está estrictamente prohibida y será amonestada con nota 1.0 para cada grupo que la realice.

Entrega:

El proyecto debe ser subido en un único archivo .zip al Aula Virtual, en el recurso respectivo, con el siguiente formato:

`Certamen-2-Grupo-ApellidosIntegrantes.zip`
Ejemplo: `Certamen-2-PerezGonzalezTapia.zip`

Dentro de este archivo deben incluir:

- Archivo(s) fuente(s) bien documentado(s).
- Archivo Makefile para realizar la compilación y cualquier elemento que facilite la comprensión de sus programas.
- Archivo Readme.txt con todas las instrucciones necesarias para la correcta ejecución del programa.

Sistema de Evaluación:

La revisión será en vivo, todos los integrantes del grupo deben estar presentes.
Además, cada estudiante debe responder una encuesta que se le hará llegar, indicando el factor de participación (FP2) para cada compañero/a de grupo. La nota del Certamen 2 (C2) estará dada por una

nota grupal (NG2) resultante de los programas implementados, y un factor de participación (FP2) entregado por sus pares de forma anónima: Nota C2 = NotaGrupal2 * FP2

Nota Grupal:

Criterio	Puntos
Calidad del código: Los códigos implementados se ejecutan sin errores; son claros, está bien documentados y se ajustan a las condiciones de entrega. Los datos de entrada y salida están comentados en el archivo readme.txt.	10
Problema 1 Problema 2	35 55
Total	100

Factor de Participación:

La nota FP2 para un/a estudiante corresponde al promedio de los factores ingresados por sus compañeros/as. Eventualmente, también pueden incidir las respuestas a preguntas realizadas por el profesor.

Nivel de Participación	**Factor**
Excelente: Mi compañero/a fue indispensable para realizar del trabajo. Sin él/ella no se podría haber realizado (***solo puede evaluar a máximo 1 integrante de esta manera***).	**1.1**
Bueno: Mi compañero/a se desempeñó de manera satisfactoria y responsable en el equipo.	**1.0**
Insuficiente: Mi compañero/a no participó lo suficientemente en el trabajo. Su desempeño no fue el ideal esperado.	**0.8**
Deficiente: Mi compañero/a no fue un aporte al equipo. El equipo habría funcionado mejor sin él/ella.	**0.6**