OPERATING SYSTEMS

2024-2025

SIMULATOR OF A MULTIPROGRAMMED COMPUTER SYSTEM with clock interrupts and a new Long Term Scheduler
V3

Introduction

We are going to modify the Long Term Scheduler we have since version V2, in order to allow programs to arrive the system in any time (until now, we supposed all programs arrived at instant 0).

There are several changes in different parts of the simulator. Here we describe the most important ones.

Design

The Computer System

- New or modified data structures:
  - Program arrival time queue (`arrivalTimeQueue`), implemented as a binary heap, using arrival time as ordering criteria. Its management is done in a similar way as the other heaps used in the system.

The Operating System

- Functionality:
  - Long Term Scheduler (LTS).
    - It will be invoked at system startup and with every clock interrupt.
  - Interrupt handling routines.
    - We will modify the clock interrupt handling routine, to invoke the LTS.
  - Process table entries occupied by zombie processes are released, when necessary.
  - It is possible for the simulation to finish while the System Idle Process is executing.

SIMULATOR OF A MULTIPROGRAMMED COMPUTER SYSTEM with clock interrupts and a new Long Term Scheduler

V3

Initial tasks

Create a copy of your `V2` directory (with all the exercises finished) and rename it as `V3`. Then, copy the contents of `V3-studentsCode` (eCampus) into your `V3` directory. Finally, execute the following command in your `V3` directory:

```
$ make clean
```

The following exercises must be done working with the contents of the copied set of files.

Exercises

We are going to allow processes to appear in the system at any time. Starting this version, we will specify the arrival time of each process in the command line (as if we were clicking at that moment in the corresponding icon in Windows):

```
./Simulator ex1 4 ex2 5 ex3 7 ex4 0
```

The arrival time may also be specified inside the optional files that can be used to set a list of (user or daemon) programs to execute. Just add the arrival time after the program name, separated by a space or a comma.

To implement this new functionality, it is necessary to use a program arrival queue, implemented as a heap:

1. Implementation and use of the program arrival queue.

   a. Paste the following lines of in `ComputerSystem.c`:

   ```
   heapItem *arrivalTimeQueue;
   int numberOfProgramsInArrivalTimeQueue=0;
   ```

   and create the heap the same way you did with the rest of heaps in the system, with a size equal to the programs list, and before initializing the operating system.

   b. We need additional functionality in the Computer System component to manage the program arrival queue. That functionality is provided in `ComputerSystemBase.c` and `ComputerSystemBase.h`, so you only have to add to `ComputerSystem.h` the following statement:

   ```
   #define ARRIVALQUEUE
   ```

After that, you should be able to compile all the source code files of the Simulator (specially `ComputerSystemBase` and `Heap`).

c. The function `ComputerSystem_FillInArrivalTimeQueue()` in `ComputerSystemBase.c` is the responsible of adding user programs to the program arrival queue (the heap), sorted by the arrival time.

Invoke that function **immediately before the invocation to the Long-Term Scheduler**, during the operating system initialization.

d. Add an invocation to `OperatingSystem_PrintStatus()` **just after the invocation to the function mentioned in previous paragraph** (exercise 1.c), during operating system initialization.

2. In `OperatingSystemBase.c` there is a new function, `OperatingSystem_IsThereANewProgram()`. That function returns `YES` (already existing constant) if there exists at least one user program whose arrival time is less than or equal to the current clock tic and has not been considered yet by the LTS. If it does not exist, it returns `NO`; and, if there are no more user programs in `arrivalTimeQueue`, it returns `EMPTYQUEUE`. For example, if we execute the simulator this way:

```
./Simulator A ex1 9 ex2 8 ex3 14 ex4 0
```

and then, call the function `OperatingSystem_IsThereANewProgram` when the clock value is 12, the function would return `YES`, because programs `ex1` and `ex2` arrival times are less than 12, program `ex3` will arrive in the future and `ex4` has already been considered by the LTS (remember the default value of `INTERVALBETWEENINTERRUPTS` is 5).

Using this new function:

Modify your Long-term Scheduler so it only continues trying to create processes while **repeated invocations** to `OperatingSystem_IsThereANewProgram` return `YES`, that is, there is at least one additional program arriving so far. To extract the program from the `arrivalTimeQueue`, use the `Heap_poll()` function (revise how it is invoked).

If the exercise is correctly solved, the simulator will only create and execute programs with arrival time equal to zero.

3. In the clock interrupt handling routine:

a. Add a call to your long-term scheduler in your clock interrupt handling routine. **Put it after the code that deals with awaking processes**. In this way, the long-term scheduler will be executed:

i. During operating system initialization and

ii. whenever a clock interrupt occurs.

Remember, if a process was created, the `OperatingSystem_PrintStatus()` function must be invoked (exercise 5d, V2).

b. If the **Long Term Scheduler** successfully created new processes or any number of processes woke up, it will be necessary to check if the executing one is still the most suitable to use the processor among all those competing for it. If it is not, it will be necessary to replace the

executing process with the one with the highest priority and show a message like the following (`SHORTTERMSCHEDULE` section):

```
[27] Process [1 - prNam1] is thrown out of the processor by process [2 - prNam2]
```

An invocation to `OperatingSystem_PrintStatus()` must take place in this case.

Notice that the behaviour described in the preceding paragraphs is almost identical to what was implemented in V2 for unblocked processes (but, including the processes that may be created by the LTS).

4. In order to avoid possible infinite loops, check (and modify your solution, if necessary):
    a. The condition that stops the simulation just after the long-term scheduler has executed during the operating system initialization:
        i. Perhaps, no process has been created in time 0 but some other programs have a greater arrival time.
    b. The condition that stops the simulation after the execution of the long-term scheduler when a clock interrupt is handled.
    c. The condition that stops the simulation each time a process finishes its execution.

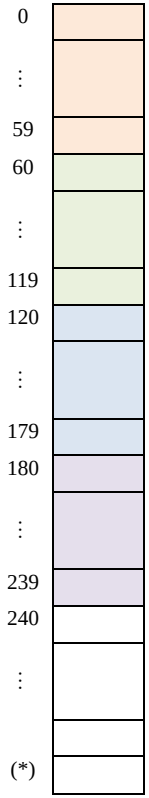
School of Computer Science. Operating Systems: Simulator V3

MainMemory

MAR

MBR

mainMemory



(*) = MAXMEMORYSIZE - 1

Clock

tics

MMU

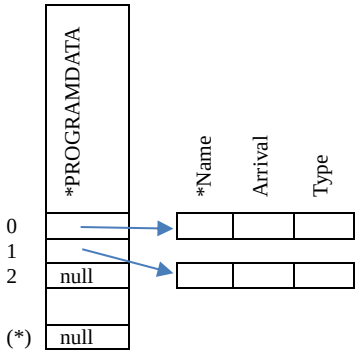
Base

Limit

MAR

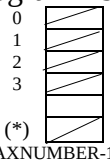
ComputerSystem

ProgramList



(*) = PROGRAMMAXNUMBER-1

Arrival
(index in programList)



(*) = PROGRAMMAXNUMBER-1

NumArrival

Messages Subsystem
…

Processor

accum

PC

IR

MAR

MBR

PSW

A
B
C
D

SP

Int

VInt.

0	null
1	null
2	SysCall Entry
3	null
4	null
5	null
6	Exception Entry
7	null
8	null
9	ClockInt Entry
…	…
(*)	null

(*) = INTERRUPTTYPES -1

OperatingSystem

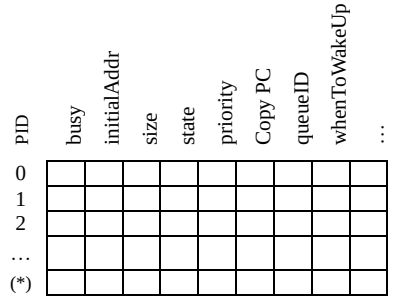
executingProcessID

sipID

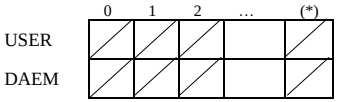
NonTerminated

numberOfClockInterrupts

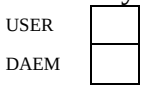
ProcessTable



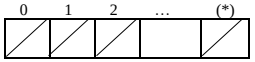
ReadyToRun (PID)



NumReadyToRun



Sleeping (PID)



NumSleeping

(*) PROCESSTABLEMAXSIZE - 1