Laboratory assignment 2 in Operating Systems

OSP: An Environment for Operating System Projects

*Lab*2 is based on *OSP:An Environment for Operating System Projects*. Before you start with the lab, you need to read the paper: *Introduction to OSP* and answer the *Questionnaire for Laboratory assignment 2*. After submitting the answers to the Fire system, you will get an initialization command required for completion of this lab.

The lab is divided in two sub-projects. In each project you will program a module of an operating system in the C language and test the code with a simulator that is included in the OSP system. A demo version of OSP is available under the name:

/chalmers/groups/cab_ ce_ edu_ 2010_ eda092_ os_ -/OSP/lab2.1.linux/OSP.demo

Observe that OSP gives different printouts in the different projects.

Sub-project 1 - CPU Scheduling

In this project you will implement the CPU module of OSP. Read chapter 1.6 (CPU Scheduling) in the OSP paper before starting. Start by executing the initialization command you will get after submitting the answers to the preparations questionnaire to the Fire-system. This command will create the directory lab2.1. The directory contains the files *Makefile, cpu.c, dialog.c, par.debug*.

In cpu.c you shall implement a preemptive Round Robin scheduler. The file cpu.c already contains the needed external declarations. You may not change the given declarations but you may add your own declarations as needed. Write the code for the subroutines *cpu_ init()*, *dispatch()* and *insert_ ready(pcb)*. You may add your own subroutines as needed. The file *dialog.c* contains routines that may be useful for debugging your code. The program is compiled and linked by the UNIX command: *make*. The result of the compilation is an executable file named OSP. The parameter file *par.debug* can be used for a first test. This first sub-project is intended as a rather simple introduction to OSP. Make the code as simple as possible. There is no other performance requirements in this sub-project than that the code shall work correctly.

Submitting the project

The OSP system includes a hand_in command that shall be used to submit the projects. To submit the project you execute the command:

/chalmers/groups/cab_ce_edu_2010_eda092_os_-/OSP/lab2.1.linux/hand_in

The hand_in command asks for the names of the students and which parameter files to use. The hand_in command looks for the parameter files in the directory:

/chalmers/groups/cab_ce_edu_2010_eda092_os_-/OSP/lab2.1.linux

If the files are not found they will be looked for in current directory. The following parameter files shall be used when submitting sub-project 1:

- par.trace
- par.low
- par.high

The first file makes a short simulation with the trace switch on. The second file tests your program with a low frequency of new processes. In par.high the frequency of new processes is high. A copy of the simulation is placed in the file *simulation.run* in the current directory. Check that the content of *simulation.run* looks correct after you have done the submission. An assignment can be submitted more than once, because hand_in will replace an earlier submission. The source code file should include the names of the group members and the group number.

Some Hints for Lab 2.1

- You will use a linked list to manage the ready queue. Students who have attended the exercise lection "Discussion on 2nd prg-assignment" in course week 5 will have a linked list they can adapt to the lab. Other students are free to use any linked list code from elementary algorithm books (as long as they have a reference).
- Make sure the linked list works BEFORE inserting it into OSP (debugging inside OSP is not straight-forward!). We will not help for linked list related problems (typically segmentation faults)
- Read OSP manual carefully and MARK ACTIONS (e.g. change status field -> ready, etc.)
- Insert the code directly into cpu.c
- You must set set_time(quantum) in dispatch().
- The ready queue constists of processes READY to run ONLY!
- No need to prepage!
- When submitting, you must use the order of the parameter files as specified in this lab PM!!
- After submission according to this PM, also submit cpu.c to the fire-system (for administrative purposes)

Sub-project 2 - Memory Management

In this project you will implement the MEMORY and PAGEINT modules of OSP. Read chapter 1.4.3 (page fault handling) and chapter 1.5 (memory management) in the OSP paper before you start. Start by executing the initialization command you will get after submitting the answers to the preparations questionnaire to the Fire-system. This command will create the directory lab2.2. The directory contains the files *Makefile, memory.c, pageint.c, dialog.c and par.debug.*

In the MEMORY module the goal should be to get as low page fault frequency as possible. Other parameters that also should have as low value as possible are average waiting time and average turnaround time.

You shall implement a clock algorithm similar to the one used in older UNIX systems. The principle for the algorithm is described in the course book (called second-chance) and in the 4BSD article. Some additional details are given here. When the number of free page frames falls below some small value *min_free* (for example 10% of all page frames, in our case 4 page frames) a special *page_ daemon* process starts. In OSP the *get_page* routine is the best place to call the *page_ daemon* routine. When the page_daemon is started it will scan a number of frames in the frame table until it ran long enough so that a specified number of page frames has been freed *(lots_free)*. The *page_ daemon* will reset the references bit in the page-table if it is set. If the reference bit already was reset, the page will be set free. (Do not forget to write the page to disk if it was modified).

Submitting the project

To submit the project you execute the command:

/chalmers/groups/cab_ ce_ edu_ 2010_ eda092_ os_ -/OSP/lab2.2.linux/hand_ in

The following parameter files shall be used when submitting sub-project 2:

- par.trace
- par.low
- par.high

The first file makes a short simulation with the trace switch on. The second file tests your program with a low frequency of new processes. In par.high the frequency of new processes is high.

The source code file shall include the names of the group members. Furthermore the source code shall contain an explanation of your strategy and your statistics (max one page).

Your explanation should include at least the following points:

Strategy:

Based on the resulting statistics, motivate in a few sentences your (optimal) choice of:

1. Condition for start of page_daemon (value of *min_free*)
2. How many pages the page_daemon will set free each time (value of *lots_free*)

Statistics:

Investigate how the page fault frequency varies as a function of *min_free* and *lots_free.* (The page_daemon is started when the number of free pages is less than *min_free* until *lots_free* pages have been freed). Do this by running OSP serveral times with par.high while varying the numbers of *min_free* and *lots_free.* Write down the results in a (ascii) table (no absolute values!!! use normalized pagefault frequency = pagefaults/memory references) and add it to the stratagy discussing at the beginning of your sourcecode.

Some hints for Lab2.2

- Familiarize yourself with the code (esp. structure of page table entry and frame table entry)
  - Write structures and fields (and how structures are connected) on a piece of paper to have the easily available.
  - Always give ALL fields a reasonable value in all functions you write. If you swap out a page, set it that it is no longer valid.
  - Never, never reset *frame_id* to -1 (set only in *get_page*)
  - A reasonable value for *page_id* for an unused page is -1. This way, OSP will print nice ~ in its output.
  - A reasonable value for an unused pcb is NULL.
  - Never modify *lock_count* in frames.
- There is an error routine to print contents of the page table. Use it to debug and learn: *print_page_tbl().*
- If you create possilby infinite loops, have some error code that will warn for this as your logic might fail.
- Some pages will ALWAYS be locked for I/O. Do not assume you can have all free.
- When submitting, you must use the order of the parameter files as specified in this lab PM!!
- Make sure to remove ALL debugging code (like printfs) from your code when submitting!
- After submission according to this PM, also submit memory.c to the fire-system (for administrative purposes)