OS-344 Assignment-1

Instructions

- **This assignment has to be done in a group.**
- **Group members should ensure that one member submits the completed assignment within the deadline.**
- **Each group should submit a report, with relevant screenshots/ necessary data and findings related to the assignments.**
- **We expect a sincere and fair effort from your side. All submissions will be checked for plagiarism through a software and plagiarised submissions will be penalised heavily, irrespective of the source and copy.**
- **There will be a viva associated with assignment. Attendance of all group members is mandatory.**
- **Assignment code, report and viva all will be considered for grading.**
- **Early start is recommended, any extension to complete the assignment will not be given.**

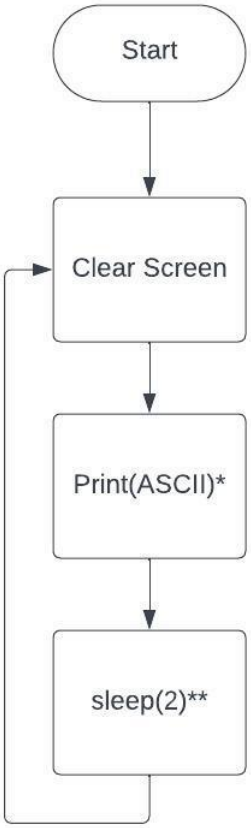
Task 1.1: Sleep

Implement a user-level sleep program for xv6, along the lines of the UNIX sleep command. Your sleep should pause for a user-specified number of ticks. A tick is a notion of time defined by the xv6 kernel, namely the time between two interrupts from the timer chip. Your solution should be in the file user/sleep.c.

Hints

- Before you start coding, read Chapter 1 of the xv6 book
- Put your code in user/sleep.c. Look at some of the other programs in user/(e.g.,user/echo.c,user/grep.c, and user/rm.c) to see how command-line arguments are passed to a program.
- Add your sleep program to UPROGS in Makefile; once you've done that, make qemu will compile compile your program and you'll be able to run it from the xv6 shell.
- If the user forgets to pass an argument, sleep should print an error message.
- The command-line argument is passed as a string; you can convert it to an integer using atoi (see user/ulib.c).
- Use the system call sleep. You can read up the implementation of wait() system call.
- See kernel/sysproc.c for the xv6 kernel code that implements the sleep system call (look for sys_sleep), user/user.h for the C definition of sleep callable from a user program, and user/usys.S for the assembler code that jumps from user code into the kernel for sleep.
- sleep's main should call exit(0) when it is done.
- Look at Kernighan and Ritchie's book *The C programming language (second edition)* (K&R) to learn about C.

Task 1.2: User Program to display animation:



* Text / Image of your choice

** seconds can be varied for faster or slower

Similar to the sleep utility, create User program to implement the above state diagram. You can be creative with the text or image to be displayed, an animation should pop up when the program is invoked. You can try with different values of seconds to sleep to get a good effect.

When the OS runs, your program's binary should be included in fs.img and listed if someone runs ls at the xv6 shell's command prompt.

Task 1.3: Statistics

In future tasks you will implement various scheduling policies. However, before that, we will implement an infrastructure that will allow us to examine how these policies affect performance under different evaluation metrics.

The first step is to extend the proc struct (see **proc.h**). Extend the proc struct by adding the following fields to it: **ctime**, **stime**, **retime** and **rutime**. These will respectively represent the creation time and the time the process was at one of following states: SLEEPING, READY(RUNNABLE) and RUNNING.

➢ **Tip:** These fields retain sufficient information to calculate the turnaround time and waiting time of each process.

Upon the creation of a new process the kernel will update the process's creation time. The fields (for each process state) should be updated for all processes whenever a clock tick occurs (you can assume that the process' state is SLEEPING only when the process is waiting for I/O). Finally, care should be taken in marking the termination time of the process (note: a process may stay in the 'ZOMBIE' state for an arbitrary length of time. Naturally this should not affect the process' turnaround time, wait time, etc.). Since all this information is retained by the kernel, we are left with the question of extracting this information and presenting it to the user. To do so, create a new system call wait2 which extends the wait system call:

int wait2(int *retime, int *rutime, int *stime)

Input:

int * retime / *rutime / *stime - pointers to an integer in which wait2 will assign:

***retime** : The aggregated number of clock ticks during which the process **waited**

***rutime** : The aggregated number of clock ticks during which the process was **running**

***stime** : The aggregated number of clock ticks during which the process was **waiting for I/O** (was not able to run).

Output:

Pid - of the terminated child process - if successful

1 - upon failure

Submission instructions

- Have the code ready to be verified during evaluation.
- Create a patch of your modified files.
- The report.pdf should contain a detailed description of all of your implementation including screenshots of code and output. This is to be uploaded the day before verification within the deadline.
- Further, you must describe your test cases in some detail, and the observations you made from them.

--End of Assignment-1--