School of Computing and Information Systems

COMP30023: Computer Systems

Tutorial Week 3

User Level thread: The OS does not know about threads in one process. User Level Threads exist when the process is executed in the user level mode.

Kernel Level thread: The OS knows about threads and threads are directly managed by the OS. Kernel Level thread are created when the system call is made for process to be executed in the kernel mode.

Process lifetime, scheduling, interprocess communication and threads

1. In the lectures, a multi-threaded text editor was shown. If the only way to read from a file is the normal blocking read system call, do you think user-level threads or kernel-level threads are being used for the text editor? Why?

   Kernel Level Thread.

2. What is a race condition? What is a deadlock?

   Dead lock is a situtation in which one process is waiting for another process to release the shared object for it to work on. However, the second process is also waiting for the first one to release another shared object.

3. In the lectures we saw that the priority inversion problem can happen with processes. Can the priority inversion problem happen with user-level threads? Why or why not?

   Yes because user-level threads in one process still need to share the same address space in that process.

4. In lectures we saw a concept of a resource graph and a circular chain that leads to a deadlock. Give an example to show that the set of processes deadlocked can include processes that are not in a circular chain in the corresponding resource allocation graph. Recall that processes and resources are represented as nodes and there is a directed edge from a process to a resource that it needs to acquire and an edge from a resource to a process that indicated that this process has acquired this resource.

5. Can a measure of whether a process is likely to be CPU bound or I/O bound be determined by analysing source code? How can this be determined at run time?

Continued on the next page

Weekly tutorial participation activity

To obtain a weekly tutorial mark (1% of your overall mark for the subject, totalling to 10% over the semester), please answer the following questions in Canvas Quiz (called **Week 3 Tutorial Activity**). You can have multiple attempts but need to submit the quiz by **11:59 pm AEDT on the day of your tutorial**. Only answering all questions correctly will give you the week's mark.

During the tutorial, your tutor will provide you with the *access code* that will unlock the quiz for the corresponding week. The access code is valid only for students in this tutorial. As a result, we expect you to attend your tutorial class as otherwise the access code you obtain in another tutorial will not work for you.

If you have a valid reason for not being able to attend your tutorial, please fill in the form accessible via Canvas by Friday 8pm AEDT of the corresponding week. You will be given an access code during the next business day. *The code will not be provided otherwise, hence, please do not email the subject coordinator or other staff members including your tutor asking for the code.* We will monitor such requests and may limit the number of times an access code is given to the same student throughout the semester to encourage participation and attendance of the tutorials.

1. Consider the following piece of C code:

```
void main( ) {
  fork( );
  fork( );
  exit( );
}
```

User fork system call():

- total number of process: 2^n
- n is the number of the fork() commands
- we have in our code.

How many child processes are created upon execution?

(a) 0
(b) 1
(c) 2
(d) 3
(e) 4

2. Choose objects that are shared between threads (Multiple answer question):

(a) Program code
(b) Global variables
(c) Local variables

(d) Dynamically allocated memory

(e) Registers

3. Select the number of processes that can be deadlocked with each other (Multiple answer question):

   (a) 2

   (b) 3

   (c) 4

   (d) Any number involving more than 2 processes

4. Suppose a scheduling algorithm operates as follows. If a new process gets created while another process is running, the scheduler suspends it and gives priority to the new process by allocating it to the CPU. This scheduling algorithm is preemptive.

   (a) True

   (b) False