SINGAPORE POLYTECHNIC
SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING
ET0023 OPERATING SYSTEMS

TUTORIAL 4 Threads – Suggested Solutions

1.

Programs	Processes	Threads
Code – usually textual, Compiled or interpreted into machine code	Machine Language (Binary)	Machine Language (Binary)
Saved in a file on disk	Loaded in memory, ready to execute or executing	Part of a process, a function within the process
Needs to be loaded into memory by the Process manager	Requires a Process Control Block which defines code, data, heap and stack. Also needs to define storage and I/O access	Has own ID, Program Counters, Registers, Stack, shares memory and access with other threads from the same PCB
Created using text editor, compiled or interpreted into machine code	Machine code from program, loaded and run by dispatcher	Created by the User to run in parallel with other threads within the same PCB

2. Information that is stored by the thread

Name/ID	For identification and ownership by process
Program Counter	To keep track of the execution of code within thread
Register set	Own register set for computation and execution of code
Stack Space	Temporary storage for variables etc.

The Data and Address space is shared by the threads created by the PCB.
A thread is considered light-weight as only registers (incl id, stack space etc) are all in memory and a context switch by the CPU would only involve moving from the CPU registers and memory which is very fast considering the size of the registers and the data involved.

3. Cache Hit – time take to retrieve data from memory = 15 msec (fast)
Cache Miss – time to retrieve data from disk (data is not in memory) and as you have to check the memory first before you access the disk, time taken = 15 + 75 = 90msec.

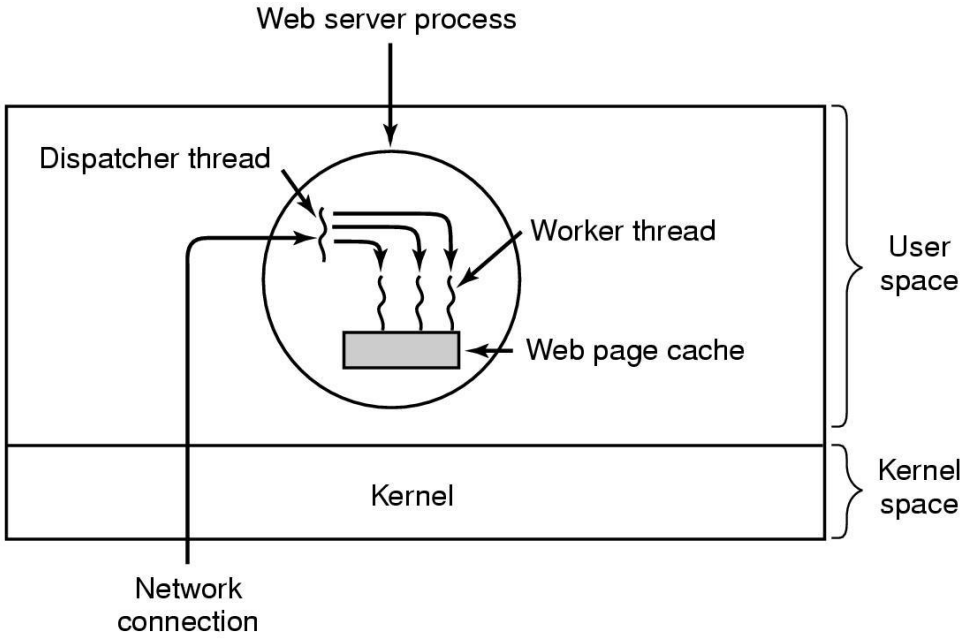
Assuming only single-threaded (or single process) server, then
2/3 of the accesses are Cache Hits
1/3 of the accesses are Cache Misses
Hence average access time = (1/3 * Time for Cache Miss) + (2/3 * Time for Cache Hit)
= (1/3 * 90) + (2/3 * 15) msec
= 30 + 10 msec
= 40 msec.
Hence, for single threaded server, number of accesses/sec = 1.0 sec / 40
= 1000 / 40
= 25 access/sec

However, for a multi-threaded server, the Cache Hit and the Cache Miss are 2 independent operations running in parallel. While on thread obtains a Cache Hit and retrieves the data from the memory, the thread with the Cache Miss can be accessing the disk

Hence, average access time = 15 msec
Hence, for multi-threaded server, number of accessec/sec = 1.0 sec / 15
= 1000 / 15
= 66.7 access/sec

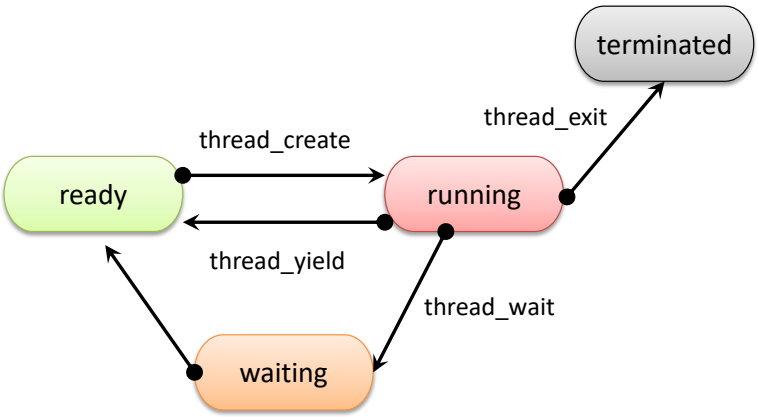
4. A multi-programming environment uses multiple CPUs or a CPU with multiple CORE or a CPU able to handle multiple threads. Hence, the threads are able to run in parallel.
Advantages:
   i. Allows the programmer to write programs that allow switching between lightweight processes when it is best for the program.
   ii. Increases responsiveness
   iii. Allows sharing of resources from the PCB, resulting in low-weight context switchin
   iv. Economical as switching threads has low overhead.
   v. Utilization of the CPU architecture

A multi-threaded HTTP Server (ref: Tenanbaum Chpt 2 pp88-89 found in BlackBoard, Lect04)



5. The functions are used by the Programmer to create threads to run/switch between processes when it is best for the program.

Thread_create	Creates a thread, giving it an ID and name of the procedure/code to run
Thread_exit	A signal given by the thread that it has finished and is no longer schedulable
Thread_wait	A signal given by a thread that it is waiting for another thread to exit before continuing
Thread_yield	A thread voluntarily gives up the CPU to let another thread run



6. All 3 functions are used within programs (C/C++) in the Linux system to create processes. The difference lies in the way the processes are created, hence, giving rise to the type of OS process creation intended.

exec()
The exec() family of functions will initiate a program from within a program. You would use these functions to run a program within another program using either current environment variables or parameters passed to the program.

fork()
The fork() system call will spawn a **new child process** which is an identical process to the parent except that has a new system process ID. The process is copied in memory from the parent and a new process structure is assigned by the kernel. The return value of the function is which discriminates the two threads of execution. A zero is returned by the fork function in the child's process.
The environment, resource limits, umask, controlling terminal, current working directory, root directory, signal masks and other process resources are also duplicated from the parent in the forked child process.

clone()
The function clone() creates a new child process which shares memory, file descriptors and signal handlers with the parent. It implements **threads** and thus launches a function as a child. The child terminates when the parent terminates.