UECS2103/2403/2423 Operating Systems
Tutorial 1 (Q&A)

1. User program, user data, system stack and process control block.

2. Process identification, processor state information, and process control information.

3. A blocked process is still resident in main memory whereas a suspended process is a process that has been swapped out from main memory.

4. While the process is running, it is pre-empted, then it is swapped out from the main memory and finally it is swapped into the main memory again in a later time.

5. New → Ready → Running → Ready → Running → Blocked →Blocked/Suspend →Ready/Suspend → Ready → Running → Ready →Ready/Suspend → Exit

6. a) Assign a unique identifier.
   Allocate space for the process.
   Initialise process control block.
   Set appropriate linkages.
   Create / expand other data structures.

   b) New → Ready→ Running→ Ready→ Running→ Blocked→ Blocked/Suspend→ Ready/Suspend→ Ready→ Running→ Exit

7. a) New → Ready → Ready/Suspend → Ready → Running → Ready → Running → Blocked → Exit

   b) Save context of processor.
   Update the process control block of process P.
   Move the process control block to appropriate queue.
   Select process Q to execute.
   Update the process control block of process Q.
   Update memory-management data structures.
   Restore the processor context of process Q.

UECS2103/2403/2423 Operating Systems
Tutorial 2 (Q&A)

1. User → kernel mode
   Initially, the program executes in user mode, mode change occurs when it needs to transfer photo from camera.

   Kernel → user mode
   Once all the photos transferred, mode change occurs again.

   User → kernel → user mode
   Mode is changed when the program needs to store selected photos into the flash drive. Once the I/O operation completed, execution mode changed to user mode again.

2. i. New → Ready : Long term scheduling
   ii. Ready → Running : Short term scheduling
   iii. Blocked →Blocked/Suspend : Medium term scheduling

3. When none of the processes in main memory is in the Ready state, the operating system swaps one of the blocked processes out onto disk into a suspend queue, so that another process may be brought into main memory to execute.

4. a.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
i	B	B	A	A	B	B	A	D	D	C	C	E	E	D	D	C	C	E	D	D
ii	B	B	B	A	A	A	B	D	D	D	C	C	C	E	E	E	D	D	D	C
iii	B	B	B	B	A	A	A	D	C	C	C	C	E	E	E	D	D	D	D	D
iv.	B	B	B	B	A	A	A	D	D	D	D	D	D	E	E	E	C	C	C	C

   b. The total time that a request spends in the system (waiting time plus service time).

   c.

	A	B	C	D	E
RR q = 2	5	6	9	15	9
RR q = 3	4	7	12	14	7
SRT	5	4	4	15	6
HRRN	5	4	12	8	7

UECS2103/2403/2423 Operating Systems
Tutorial 3 (Q&A)

1. Thread switching does not require kernel mode privileges.
   The scheduling algorithm can be tailored to the application.
   ULTs can run on any operating system.

2. The process will be blocked because the thread structure of a process is not visible to the kernel.

3. All the threads will be terminated as well because the threads are part of the process.

4. The thread switching within the same process requires a mode switch to the kernel.

5. a) As all the threads involve I/O operations, the tasks should be implemented as Kernel-level threads so that only the thread that issues the I/O request will be blocked.

   b) i) Process: Blocked
      Thread: Running
      The kernel doesn't aware of ULT, it will block the process.

      ii) Process: Running
      Thread: Blocked
      KLT managed by kernel, kernel will block the thread.

6. Yes. Threads can share file easily.

7.
   a) New → Ready→ Running→ Ready→ Ready/Suspend→ Ready→ Running→ Blocked→ Ready→ Exit

   b) Process K starts: user mode
      To transfer file: user mode → kernel mode
      File transfer completed: kernel mode → user mode

   c) Yes, because the kernel/OS does not know the existence of user-level thread.

8. Since all threads **communicate** with each other **without involving input** from I/O devices, it is best to be implemented as **User-level** threads.
   The creation, termination, communication and switching between user-level threads are easier and do not invoke the kernel, thus, overhead reduced.

UECS2103/2403/2423 Operating Systems
Tutorial 4 (Q&A)

1. Interrupt disabling, compare and swap instruction, exchange instruction.

2. Busy-waiting, starvation and deadlock might occur.

3. P1-P2-Q1-Q2: x = 19
   P1-Q1-P2-Q2: x = 5
   P1-Q1-Q2-P2: x = 5
   Q1 (divide by 0, process Q will be terminated): x = 7

4. (a)

```
P(){
  semWait(s);
  write_data();
  semSignal(s);
}
```

   (b) I/O device is allocated to process P2 before process P1 is ready. When process P1 is ready, it is selected for execution. However, the device is allocated to process P2 and process P1 has to wait, meanwhile, process P2 is also waiting for process P1 to finish so that it can continue. Deadlock occurs.

5.

```
sem p=1, q=1, filled=0;

server(){
  while(true){
     semWait(filled);
     semWait(q);
     get_request();
     semSignal(q);
     semWait(p);
     print();
     semSignal(p);
  }
}
```

6. Yes. The execution is switched to process Q before the second *wait* operation in process P is executed (P3).Process Q will be blocked by the second *wait* operation. When process P continues, it will be blocked by the second *wait* operation and both processes are deadlocked.

7.

```
int buf[BUF_SIZE];
int data_count=0;
semB req=0, bufsem=0, bufempty=1, buffill=0;

read_data(){
  semWait(req);
  semWait(bufempty);
  semWait(bufsem);
  //read data from data.dat into buf[]
  semSignal(bufsem);
  semSignal(buffill);
}

process_data(){
  semWait(buffill);
  semWait(bufsem);
  // get data from buf[]
  // process and update to data.dat
  // reset data_count to zero (0)
  semSignal(bufsem);
  semSignal(bufempty);
}
```