Started on	Thursday, 16 February 2023, 9:01 PM
State	Finished
Completed on	Thursday, 16 February 2023, 10:08 PM
Time taken	1 hour 6 mins
Grade	**13.65** out of 15.00 (**91.01**%)

Question **1**

Correct

Mark 1.00 out of 1.00

Which of the following statements is false ?

Select one:

- a. Response time is more predictable in preemptive systems than in non preemptive systems.
- b. Real time systems generally use non preemptive CPU scheduling. ✔
- c. Time sharing systems generally use preemptive CPU scheduling.
- d. A process scheduling algorithm is preemptive if the CPU can be forcibly removed from a process.

Your answer is correct.

The correct answer is: Real time systems generally use non preemptive CPU scheduling.

Question **2**

Correct

Mark 1.00 out of 1.00

Mark whether the concept is related to scheduling or not.

Yes	No		
◉ ☑	○ ✖	ready-queue	✔
○ ✖	◉ ☑	file-table	✔
◉ ☑	○ ✖	timer interrupt	✔
◉ ☑	○ ✖	context-switch	✔
◉ ☑	○ ✖	runnable process	✔

ready-queue: Yes
file-table: No
timer interrupt: Yes
context-switch: Yes
runnable process: Yes

Question **3**

Correct

Mark 1.00 out of 1.00

Map each signal with it's meaning

SIGALRM	Timer Signal from alarm() ✔
SIGPIPE	Broken Pipe ✔
SIGCHLD	Child Stopped or Terminated ✔
SIGSEGV	Invalid Memory Reference ✔
SIGUSR1	User Defined Signal ✔

The correct answer is: SIGALRM → Timer Signal from alarm(), SIGPIPE → Broken Pipe, SIGCHLD → Child Stopped or Terminated, SIGSEGV → Invalid Memory Reference, SIGUSR1 → User Defined Signal

Question **4**

Correct

Mark 1.00 out of 1.00

Order the sequence of events, in scheduling process P1 after process P0

context of P0 is saved in P0's PCB	3 ✔
context of P1 is loaded from P1's PCB	4 ✔
Control is passed to P1	5 ✔
timer interrupt occurs	2 ✔
Process P1 is running	6 ✔
Process P0 is running	1 ✔

Your answer is correct.

The correct answer is: context of P0 is saved in P0's PCB → 3, context of P1 is loaded from P1's PCB → 4, Control is passed to P1 → 5, timer interrupt occurs → 2, Process P1 is running → 6, Process P0 is running → 1

Question **5**

Correct

Mark 1.00 out of 1.00

Match the names of PCB structures with kernel

xv6 struct proc ✔

linux struct task_struct ✔

The correct answer is: xv6 → struct proc, linux → struct task_struct

Question **6**

Partially correct

Mark 0.75 out of 1.00

Select all the correct statements about signals

Select one or more:

- [ ] a. Signal handlers once replaced can't be restored
- [ ] b. The signal handler code runs in kernel mode of CPU
- [ ] c. SIGKILL definitely kills a process because it's code runs in kernel mode of CPU
- [x] d. Signals are delivered to a process by another process ✖
- [x] e. SIGKILL definitely kills a process because it can't be caught or ignored, and it's default action terminates the process ✔
- [x] f. Signals are delivered to a process by kernel ✔
- [x] g. The signal handler code runs in user mode of CPU ✔
- [x] h. A signal handler can be invoked asynchronously or synchronously depending on signal type ✔

Your answer is partially correct.

You have selected too many options.
The correct answers are: Signals are delivered to a process by kernel, A signal handler can be invoked asynchronously or synchronously depending on signal type, The signal handler code runs in user mode of CPU, SIGKILL definitely kills a process because it can't be caught or ignored, and it's default action terminates the process

Question **7**

Correct

Mark 1.00 out of 1.00

Select the state that is not possible after the given state, for a process:

New: Running ✔

Ready : Waiting ✔

Running: : None of these ✔

Waiting: Running ✔

Question **8**

Correct

Mark 1.00 out of 1.00

Which of the following are NOT a part of job of a typical compiler?

- [ ] a. Process the # directives in a C program
- [x] b. Check the program for logical errors✔
- [ ] c. Check the program for syntactical errors
- [x] d. Suggest alternative pieces of code that can be written✔
- [ ] e. Invoke the linker to link the function calls with their code, extern globals with their declaration
- [ ] f. Convert high level langauge code to machine code

Your answer is correct.

The correct answers are: Check the program for logical errors, Suggest alternative pieces of code that can be written

Question **9**

Correct

Mark 1.00 out of 1.00

Select the compiler's view of the process's address space, for each of the following MMU schemes:
(Assume that each scheme,e.g. paging/segmentation/etc is effectively utilised)

Paging	one continuous chunk	✔
Segmentation	many continuous chunks of variable size	✔
Relocation + Limit	one continuous chunk	✔
Segmentation, then paging	many continuous chunks of variable size	✔

Your answer is correct.

The correct answer is: Paging → one continuous chunk, Segmentation → many continuous chunks of variable size, Relocation + Limit → one continuous chunk, Segmentation, then paging → many continuous chunks of variable size

Question **10**

Partially correct

Mark 0.67 out of 1.00

Which of the following parts of a C program do not have any corresponding machine code ?

- [x] a. #directives ✔
- [ ] b. function calls
- [ ] c. global variables
- [x] d. typedefs ✔
- [ ] e. expressions
- [ ] f. pointer dereference
- [ ] g. local variable declaration

Your answer is partially correct.

You have correctly selected 2.
The correct answers are: #directives, typedefs, global variables

Question **11**

Partially correct

Mark 1.38 out of 2.00

Select all the correct statements about the state of a process.

- [x] a. A process changes from running to ready state on a timer interrupt ✔
- [ ] b. A process in ready state is ready to receive interrupts
- [x] c. A running process may terminate, or go to wait or become ready again ✔
- [x] d. Processes in the ready queue are in the ready state ✔
- [x] e. Typically, it's represented as a number in the PCB ✔
- [x] f. A process in ready state is ready to be scheduled ✔
- [ ] g. A waiting process starts running after the wait is over
- [x] h. It is not maintained in the data structures by kernel, it is only for conceptual understanding of programmers ✖
- [x] i. Changing from running state to waiting state results in "giving up the CPU" ✔
- [ ] j. A process waiting for any condition is woken up by another process only
- [ ] k. A process changes from running to ready state on a timer interrupt or any I/O wait
- [ ] l. A process can self-terminate only when it's running
- [x] m. A process that is running is not on the ready queue ✔
- [x] n. A process waiting for I/O completion is typically woken up by the particular interrupt handler code ✔

Your answer is partially correct.

You have correctly selected 8.

The correct answers are: Typically, it's represented as a number in the PCB, A process in ready state is ready to be scheduled, Processes in the ready queue are in the ready state, A process that is running is not on the ready queue, A running process may terminate, or go to wait or become ready again, A process changes from running to ready state on a timer interrupt, Changing from running state to waiting state results in "giving up the CPU", A process can self-terminate only when it's running, A process waiting for I/O completion is typically woken up by the particular interrupt handler code

Question **12**

Correct

Mark 1.00 out of 1.00

Select all the correct statements about zombie processes

Select one or more:

- [x] a. A process becomes zombie when it finishes, and remains zombie until parent calls wait() on it✔
- [ ] b. A process becomes zombie when it's parent finishes
- [ ] c. Zombie processes are harmless even if OS is up for long time
- [x] d. If the parent of a process finishes, before the process itself, then after finishing the process is typically attached to 'init' as parent✔
- [x] e. A zombie process occupies space in OS data structures✔
- [x] f. A process can become zombie if it finishes, but the parent has finished before it✔
- [ ] g. A zombie process remains zombie forever, as there is no way to clean it up
- [x] h. init() typically keeps calling wait() for zombie processes to get cleaned up✔

Your answer is correct.

The correct answers are: A process becomes zombie when it finishes, and remains zombie until parent calls wait() on it, A process can become zombie if it finishes, but the parent has finished before it, A zombie process occupies space in OS data structures, If the parent of a process finishes, before the process itself, then after finishing the process is typically attached to 'init' as parent, init() typically keeps calling wait() for zombie processes to get cleaned up

Question **13**

Correct

Mark 1.00 out of 1.00

Mark statements True/False w.r.t. change of states of a process. Note that a statement is true only if the claim and argument both are true.

Reference: The process state diagram (and your understanding of how kernel code works). Note - the diagram does not show zombie state!

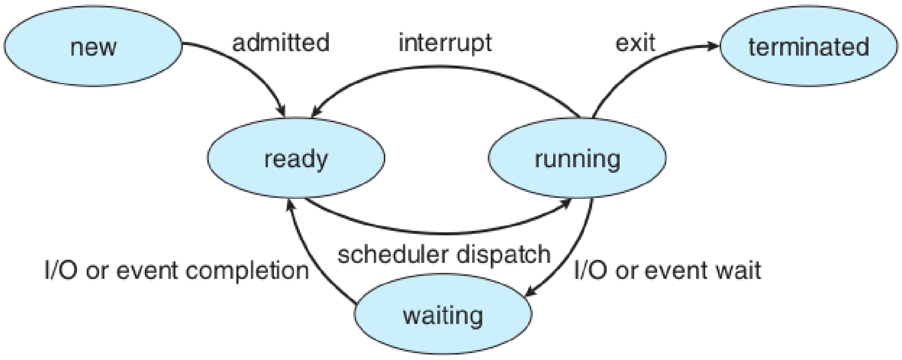

Figure 3.2 Diagram of process state.

True	False		
◉ ✔	○ ✖	A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet	✔
○ ✖	◉ ✔	A process in READY state can not go to WAITING state because the resource on which it will WAIT will not be in use when process is in READY state.	✔
◉ ✔	○ ✖	Only a process in READY state is considered by scheduler	✔
◉ ✔	○ ✖	Every forked process has to go through ZOMBIE state, at least for a small duration.	✔
○ ✖	◉ ✔	A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first	✔

A process in WAITING state can not become RUNNING because the event it's waiting for has not occurred and it has not been moved to ready queue yet: True
A process in READY state can not go to WAITING state because the resource on which it will WAIT will not be in use when process is in READY state.: False
Only a process in READY state is considered by scheduler: True
Every forked process has to go through ZOMBIE state, at least for a small duration.: True
A process only in RUNNING state can become TERMINATED because scheduler moves it to ZOMBIE state first: False

Question **14**

Partially correct

Mark 0.86 out of 1.00

Mark True/False

Statements about scheduling and scheduling algorithms

True	False			
○ ✔	◉ ✖	Generally the voluntary context switches are much more than non-voluntary context switches on a Linux system.	✖	
◉ ✔	○ ✖	A scheduling algorithm is non-premptive if it does context switch only if a process voluntarily relinquishes CPU or it terminates.	✔	
◉ ✔	○ ✖	Processor Affinity refers to memory accesses of a process being stored on cache of that processor	✔	
◉ ✔	○ ✖	xv6 code does not care about Processor Affinity	✔	
◉ ✔	○ ✖	Statistical observations tell us that most processes have large number of small CPU bursts and relatively smaller numbers of large CPU bursts.	✔	
○ ✖	◉ ✔	On Linuxes the CPU utilisation is measured as the time spent in scheduling the idle thread	✔	It's the negation of this. Time NOT spent in idle thread.
◉ ✔	○ ✖	Response time will be quite poor on non-interruptible kernels	✔	

Generally the voluntary context switches are much more than non-voluntary context switches on a Linux system.: True
A scheduling algorithm is non-premptive if it does context switch only if a process voluntarily relinquishes CPU or it terminates.: True
Processor Affinity refers to memory accesses of a process being stored on cache of that processor: True
xv6 code does not care about Processor Affinity: True
Statistical observations tell us that most processes have large number of small CPU bursts and relatively smaller numbers of large CPU bursts.: True
On Linuxes the CPU utilisation is measured as the time spent in scheduling the idle thread: False
Response time will be quite poor on non-interruptible kernels: True

◄ Random Quiz - 3 (processes, memory management, event driven kernel), compilation-linking-loading, ipc-pipes

Jump to...

Homework questions: Basics of MM, xv6 booting ►