Started on	Thursday, 16 February 2023, 9:06 PM
State	Finished
Completed on	Thursday, 16 February 2023, 10:13 PM
Time taken	1 hour 7 mins
Grade	**12.52** out of 15.00 (**83.44**%)

Question **1**

Correct

Mark 1.00 out of 1.00

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

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

Your answer is correct.

The correct answers are: #directives, typedefs, global variables

Question **2**

Partially correct

Mark 0.80 out of 1.00

Mark whether the concept is related to scheduling or not.

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

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

Question **3**

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 **4**

Correct

Mark 1.00 out of 1.00

Select all the correct statements about zombie processes

Select one or more:

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

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 **5**

Partially correct

Mark 0.67 out of 1.00

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

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

Your answer is partially correct.

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

Question **6**

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 **7**

Partially correct

Mark 0.57 out of 1.00

Mark True/False

Statements about scheduling and scheduling algorithms

True	False			
◉☑	○✖	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	✔	
◉☑	○✖	Generally the voluntary context switches are much more than non-voluntary context switches on a Linux system.	✔	
◉☑	○✖	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.
○☑	◉✖	xv6 code does not care about Processor Affinity	✖	
○☑	◉✖	Response time will be quite poor on non-interruptible kernels	✖	

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
Generally the voluntary context switches are much more than non-voluntary context switches on a Linux system.: 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
xv6 code does not care about Processor Affinity: True
Response time will be quite poor on non-interruptible kernels: True

Question **8**

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)

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

Your answer is correct.

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

Question **9**

Partially correct

Mark 0.50 out of 1.00

Select all the correct statements about signals

Select one or more:

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

Your answer is partially correct.

You have correctly selected 3.
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 **10**

Partially correct

Mark 1.38 out of 2.00

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

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

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 **11**

Partially correct

Mark 0.60 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!

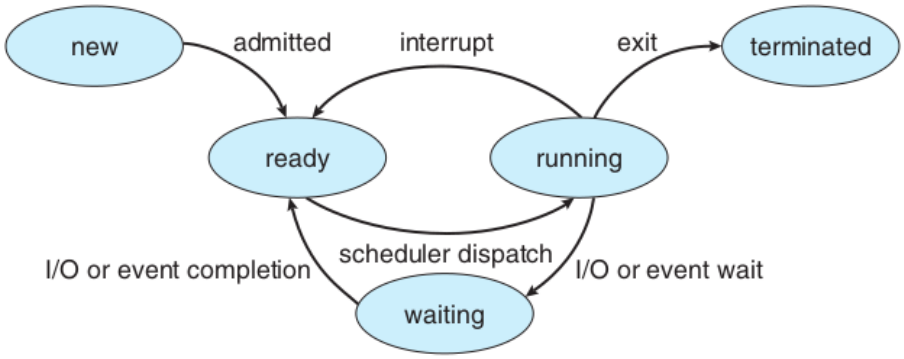

Figure 3.2 Diagram of process state.

True	False		
○ ✗	● ✓	A process 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	✓
○ ✓	● ✗	Only a process in READY state is considered by scheduler	✗
○ ✗	● ✓	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.	✓
● ✓	○ ✗	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: 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: True
Only a process in READY state is considered by scheduler: 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
Every forked process has to go through ZOMBIE state, at least for a small duration.: True

Question **12**

Correct

Mark 1.00 out of 1.00

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

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

Your answer is correct.

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

Question **13**

Correct

Mark 1.00 out of 1.00

Map each signal with it's meaning

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

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

Question **14**

Correct

Mark 1.00 out of 1.00

Which of the following statements is false ?

Select one:

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

Your answer is correct.

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

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

Jump to...

Homework questions: Basics of MM, xv6 booting ►