Tom Arad
Student ID: 11681299

What's a process?

A: A process is the execution of an image. A sequence of executions regarded by the OS kernel as a single entity for using system resources.

Each process is represented by a PROC structure.
Read the PROC structure in 3.4.1 on Page 111 and answer the following questions:

What's the meaning of:
pid, ppid?

A: pid - process ID number that identifies a process.
ppid - parent process ID number.

status?

A: the current status of the process, free/ready etc..

priority?

A: the process scheduling priority. (higher priority = more resources given by cpu).

event?

A: Event is a value which is used to decide if a sleeping process should be woken up and put into the readyQueue.

exitCode?

A: used to indicate whether a process succeeded or failed (and sometimes why).

READ 3.5.2 on Process Family Tree. What are the
PROC pointers child, sibling, parent used for?

A: The child pointer points to the 1st child of the process, the sibling pointer points to the next child of the same parent. The parent pointer points to the parent process. (this is similar to the binary tree we used in LAB 2).

fork : READ kfork() on Page 109: What does it do?

A: kfork() creates a child of the process, which is the same as the current process, and puts it inside the readyQueue.

switch : READ tswitch() on Page 108: What does it do?

Tom Arad
Student ID: 11681299

A: tswitch() writes the process's registers into the stack and then saves the address into the proc struct. If the process is still "READY" it will then enqueue it into the readyQueue with its priority. Afterwards it will dequeue the process at the start of the ready queue. It then restores the stack process using the pointer saved in the proc struct and restores the register values, and resumes execution.

exit : READ kexit() on Page 112: What does it do?

A: kexit() first deallocates all of the process's user mode context (memory and other resources), all of the process's children become P1's children. kekxit() then saves the exit value and puts the process status as "ZOMBIE". Following this kexit() wakes up the parent and also P1 if needed.

sleep : READ ksleep() on Page 111: What does it do?

A: ksleep() first saves the event for the process to wake up on. It then changes the process's status to "SLEEP" and puts it into the sleepList. Finally it gives up the cpu for the next process.

wakeup : READ kwakeup() on Page 112: What does it do?

A: when an event occurs, kwakeup() checks which processes in the sleepList are waiting on this event for waking up, it then removes those processes from the sleepList and puts them in readyQueue.

wait : READ kwait() on Page 114: What does it do?

A: kwait() checks whether the process has any "ZOMBIE" children. If so kuwait saves the pid of the zombie child as well as its exit value. It then puts the zombie child in the freeList and returns its pid. If the process has no zombie children kwait() returns -1 for error.

6. Step 1: test fork
While P1 running, enter fork: What happens?

A: P2 is removed from freeList and is enqueued into readyQueue with P1 as its parent.

Enter fork many times;
How many times can P1 fork?

A: 7

WHY?

Tom Arad
Student ID: 11681299

A: There are only 7 procs in freeList and since every time the fork command is called it uses a proc which is free, we can only call it 7 times before all the available free procs are being used.

7. Step 2: Test sleep/wakeup
Run mtx again.
While P1 running, fork a child P2;
Switch to run P2. Where did P1 go?

A: P1 was enqueued into the readyQueue.

WHY?

A: when using the switch command, tswitch() is called, tswitch() checks P1's status, and since P1's status is "READY" is puts it back into the ReadyQueue in its priority order (behind P2).

P2 running : Enter sleep, with a value, e.g.123 to let P2 SLEEP.
What happens? WHY?

A: P2 status is changed to "SLEEP", process 1 is then dequeued from the readyQueue, since it has the highest priority and resumes running. P2 is placed in the sleepList (P2 event is also given a value equal to the number we gave with the command in hexadecimal).

Now, P1 should be running. Enter wakeup with a value, e.g. 234
Did any proc wake up? WHY?

A: no procs woke up, because 234 does not match any of the event values of the procs in sleepList (currently only proc 2 who's event value is 123).

P1: Enter wakeup with 123
What happens?WHY?

A: proc 2 is enqueued into readyQueue since the wakeup value we provided was the same as proc 2's event value, prompting the system to remove it from sleepList.

8. Step 3: test child exit/parent wait

When a proc dies (exit) with a value, it becomes a ZOMBIE, wakeup its parent.
Parent may issue wait to wait for a ZOMBIE child, and frees the ZOMBIE

Run mtx;
P1: enter wait; What happens? WHY?

Tom Arad
Student ID: 11681299

A: Nothing happens, we get an error, because proc 1 has no children.

CASE 1: child exit first, parent wait later

P1: fork a child P2, switch to P2.
P2: enter exit, with a value, e.g. 123 ==> P2 will die with exitCode=123.
Which process runs now? WHY?

A: Process 1 is running now, because proc 2 was exitted, meaning it has become a zombie, so now the next running process should be the process with the highest priority in readyQueue, which was P1 in this case.

enter ps to see the proc status: P2 status = ?

A: Zombie

(P1 still running) enter wait; What happens?

A: Proc 2 has returned to the freeList.

enter ps; What happened to P2?

A: Proc 2's status is now "FREE" instead of "ZOMBIE".

CASE 2: parent wait first, child exit later

P1: enter fork to fork a child P3
P1: enter wait; What happens to P1? WHY?

A: P1 is put into the sleepList, because it is waiting on an event, which is the termination of one of its children.

P3: Enter exit with a value; What happens?

A: P3's status turns into "ZOMBIE", which wakes up P1 and enqueues it to the readyQueue. P3 is then put into the freeList with the "FREE" status (P1 wait does this), and P1 starts running because it is 1st in the readyQueue.

P1: enter ps; What's the status of P3? WHY?

A: "FREE", explained above :).

9. Step 4: test Orphans

Tom Arad
Student ID: 11681299

When a process with children dies first, all its children become orphans.
In Unix/Linux, every process (except P0) MUST have a unique parent.
So, all orphans become P1's children. Hence P1 never dies.

Run mtx again.
P1: fork child P2, Switch to P2.
P2: fork several children of its own, e.g. P3, P4, P5 (all in its childList).
P2: exit with a value.
P1 should be running WHY?

A: Because P1 is 1st in the readyQueue and it has the same priority as all the other procs.

P1: enter ps to see proc status: which proc is ZOMBIE?

A: Proc 2.

What happened to P2's children?

A: P1 became their parent.

P1: enter wait; What happens?

A: P2 is put into the freeList and its status is changed to “FREE”.

P1: enter wait again; What happens?WHY?

A: Proc 1 is put into sleepList because it is waiting for one of its children to become a zombie, and proc 3 starts running because it had the highest priority in the readyQueue.

How to let P1 READY to run again?

A: call exit once, and then call switch twice.