1. (From textbook) Systems calls provide the interface between a process and the operating systems. (I would say) A system call is a software interrupt through which application programs invoke functions provided by the underlying operating system. (Comment) Both definitions are fine. If you did not mention interrupt, however, it must show up in the comparisons with function calls.

In comparisons with function calls

- system calls are software interrupts (function calls are not)
- System calls are executed in the monitor mode, and thus have access to all privilege instructions and the entire system. Function calls are executed in the user mode.
- System call routines are part of the OS. Function call routines are part of the process (application program, shared libraries, ...)

2. There will 16 processes in total.

Analysis: We have two processes after line 1. Both of them will execute the loop in line 3 and produce equal number of children. I will focus on one of them and simply multiply the result by 2 in the end. The one I’m focusing on is called P0. Notice that we a child is created from the loop, it will continue the loop, using whatever i value it inherits from the parent.

P0, i=0 ==> P1
P0, i=1 ==> P2
P1, i=1 ==> P3
P0, i=2 ==> P4
P1, i=2 ==> P5
P2, i=2 ==> P6
P3, i=2 ==> P7

The i in all processes reaches 3 and so they terminates.

3. To wait for I/O events (mouse clicks, disk fetch completed, etc.). To wait for the completion of child processes. To wait for the arrival of messages. And so forth.

4. Please see the Gantt Charts below.

- FCFS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>P2</td>
<td>P3</td>
<td>P4</td>
</tr>
<tr>
<td>0</td>
<td>70</td>
<td>150</td>
<td>180</td>
</tr>
</tbody>
</table>

Waiting times: \( W_1 = 0, W_2 = 70 - 15, W_3 = 150 - 30, W_4 = 180 - 50. \)

Turnaround times: \( T_1 = 70, T_2 = 150 - 15, T_3 = 180 - 30, T_4 = 210 - 50. \)

- SRTF
Turnaround times: $T_1 = 130, T_2 = 210 - 15, T_3 = 60 - 30, T_4 = 90 - 50$.

- RR. I will also show the queue in order of rear, ...., front where front is the process currently having the processor.

Waiting times: $W_1 = (40 - 20) + (120 - 60) + (180 - 140), W_2 = (20 - 15) + (80 - 40) + (150 - 100) + (190 - 170), W_3 = (60 - 30) + (140 - 80), W_4 = (100 - 50) + (170 - 120)$.
Turnaround times: $T_1 = 190, T_2 = 210 - 15, T_3 = 150 - 30, T_4 = 180 - 50$.

5. This problem is, well, problematic, as many of you pointed out. I will assume the following:
12 bits to the 1st level table, 18 bits to the 2nd level tables, 10 bits to the 3rd level table. This leaves 8 bits as displacements.

(a) Page size = $2^8 = 256$ bytes.
(b) First level table size = $2^{12} \times 8$
(c) No. of third level page tables = $2^{(12+18)}$.
(d) Accesses to the first-level page table, second-level page table, and the third-level page table. Fetch the data.
(e) $50 \times 4$ ns.

6. Entities that performing the actions when interrupts occur:

- Save program counter: by processor hardware
- Lookup interrupt vector: by processor hardware
- Jump to the beginning of a service routine: by processor hardware
- Set the process state to READ: by OS (and thus software)
Set the mode bit to MONITOR: by processor hardware
Update process PCB: by OS/software

Comment: the sequence above is random and does not reflect reality. For example, the processor has to be in the MONITOR mode before executing the service routine.

7. See the discussions below.

- Forking child process: No. the parent enters the WAITING state if it has to wait for the termination of child (as the case of project 1).
- Child process terminates: Yes. If this events impacts the parent in any way, it would be to move the parent from WAITING to READY.
- IO operations: No, the process has to wait for the IO.
- IO terminates: Yes. This would move the process from WAITING to READY.
- Message receives: Yes.
- Mouse clicks. Yes.

8. Inverted page table.

(a) 8 gigabytes = 2^{33}. number of page table entries = 2^{33-12} = 2^{21}. Page table size = 2^{21} \times 6.
(b) 2^{(18+12)}

9. Linux scheduling.

(a) When all the runnable processes exhausted their time quota, counter, IO bound processes are likely in the WAITING state and thus have some of their time quota left. They are allowed to keep half of such left-overs and therefore will have more quota and higher priorities when they they become runnable.

(b) Since the scheduler chooses the highest (counter-nice) value, a high priority task will lose some of its advantages every time it is chosen, owing to a smaller counter. It will eventually yields to low priority tasks.