Homework 3 Part 2

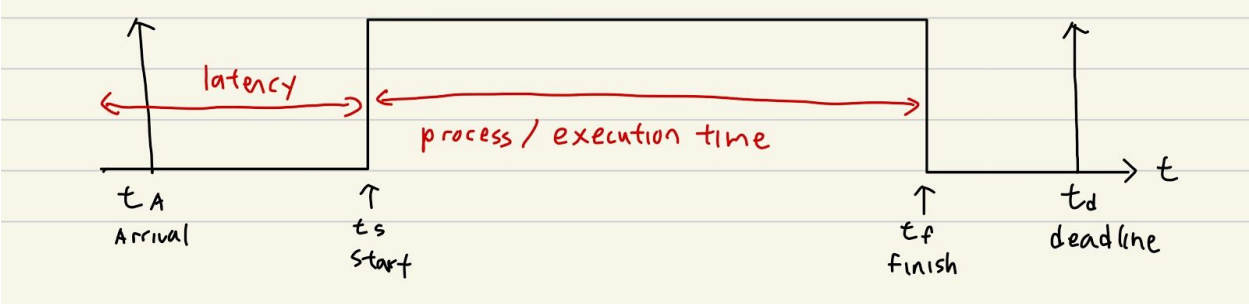
1. What are the two requirements for successful completion of an application running in a Real-Time environment?
2. Describe two processor innovations which, although they create a more efficient general-purpose computing platform, would impact the correct evolution of applications in a real-time environment. Describe how these innovations operate and what aspects of the innovations interfere with real-time applications. For each of the innovations, how might they be adapted for use with a real-time system?
3. During investigation into embedded versus real-time systems, we have been discussing typical timing examples for processes using the parameters t_a, arrival time, t_s, start time, t_f, finish time, and t_d, deadline time. Given this process timeline, what tools can we deploy to measure process execution time and process latency? Please include a timing diagram for a typical process, include the timing parameters discussed above, indicate process execution time and startup latency time on the diagram and discuss the tools used to measure each section.
4. Describe two of the four Linux kernel functions outlined in class. Figures could be helpful when describing these functions.
5. Describe the boot sequence of the Raspberry Pi in detail. What hardware modules are activated at each boot step? What software is used to boot the RPi? Where does each software process run? Identify the name and location of files used to boot the RPi. One of these files, start.elf, is an example of the ELF file format; What is the ELF format?

1. The two requirements are that an application would need to meet the individual timing requirement of each application/task and that the result must be correct.

2. The first innovation is cache. There is an instruction cache or data cache, but we will be discussing primarily on data cache. So the processor usually can access data in the RAM memory. However, what the OS will do is that if the cache is available, and if you access a specific data often in the RAM, it will end up in the data cache. The data cache is smaller and faster, and is used to store the data that is accessed repeatedly. The problem is that there is an issue called program locality. There will be data that will be accessed in cache and others will be accessed in the RAM. This causes variability or unbounded delays for the memory, because we cannot estimate exactly the time required to access a data. For example, if you have set that it needs to access the cache, but then the data is not there and you ended up accessing the RAM (this is called cache miss), this will cause issues. Also, since Linux is multi-process, when a new process comes in the cache will need to be cleared. The solution to this for real-time is to just turn off cache so that we can get the right timing, despite being slower.
The second innovation is the interrupt. Interrupt is the response by the processor of an event that needs immediate attention. It is usually of a higher priority and so if an interrupt is called, the processor will stop the current process and then go and perform the process related to the

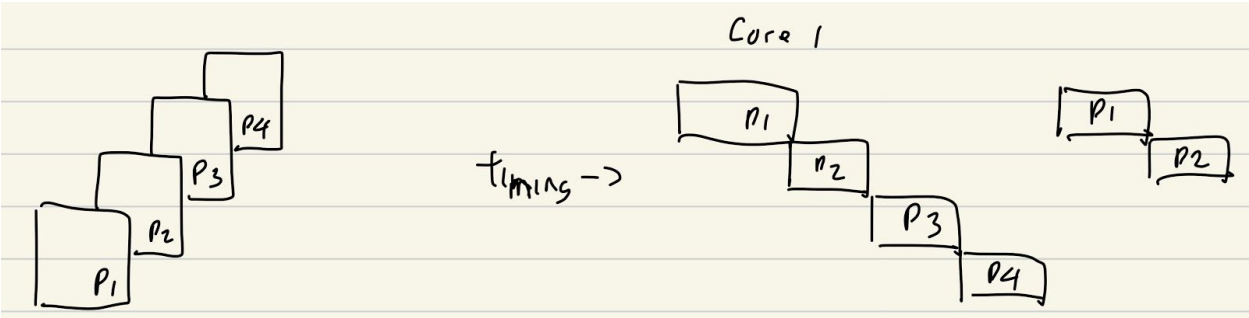
interrupt. Once it is done, it will come back to the previous process. However, an interrupt will interfere with the real-time system. So for example, if we are running an application or background process, then an interrupt is called, since it is of a higher priority the related task will be executed. If the interrupts are taking a long time, then it will stay in that process for a while and we will have an unbounded delay. This is an issue for real-time systems. So the easiest solution is to just use polling instead of interrupts, although it is not efficient. Another thing is to have the interrupts enabled, but each interrupt does not do any processing. They will run some process/interrupt handler related to the interrupt, where it will have the same priority as the main process. So, it can be handled by the OS process scheduler and manage time better, which is essential for a real-time system.

3. Below is a drawing of the process timeline.



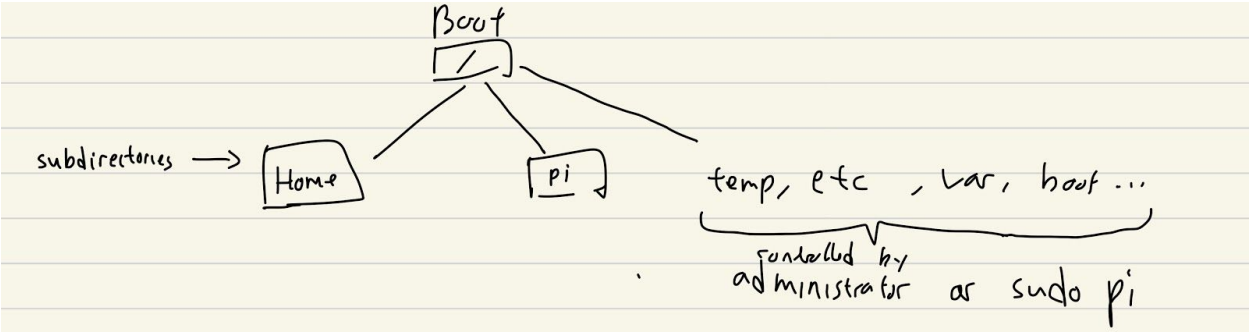
The arrival time, ta, shows when a process arrives to be scheduled. Then, when the processor starts executing the process, it is the start time ts. The time interval between ta and ts is the process latency. This can be measured on the RPi by using the cyclictest. The tool will record the ta and ts, and provide measurements of the average, max, and min latencies timings through the number of cycles of our choosing. The output can be viewed as a histogram. The next timing is the tf, which is the finish time. This marks when the process execution ends. The interval between ts and tf is the execution time, which shows how long the process is executed. This can be measured with the perf stat. Perf stat will show plenty of information/data, and one of it would be how long the process takes. The last entry is td, which is the deadline time. This is the deadline for the process, where in the real-time system it is the limit or guaranteed time in which the process will end.

4. One of the kernel functions is the process management. A diagram is included below to provide a better explanation.

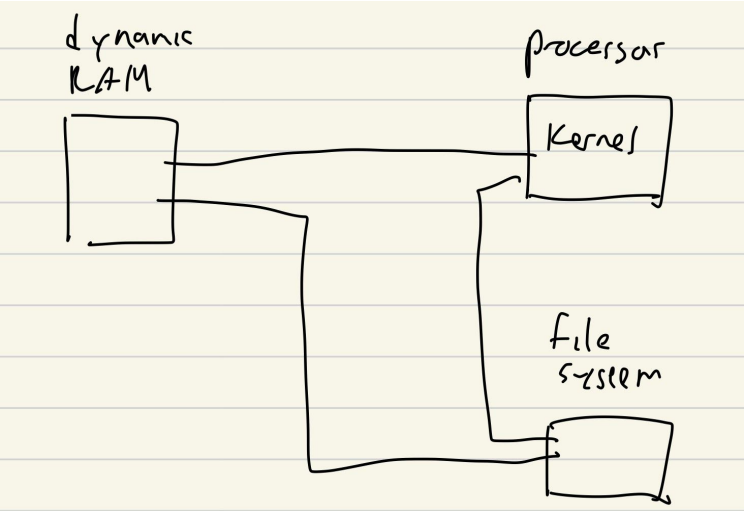


So on the left, we can see the four processes that would like to be executed. The timing diagram on the right shows how (in this example using a single core) the process will be executed. They will be executed one after the other based on a schedule. The process scheduler will allocate time slices for the multiple processes. Humans will see as if they are running simultaneously because it is very fast, but each process actually has their own schedule. The scheduler that is used is the Round Robin scheduler, in which they will implement time slices, decide on the run order, and use only one priority level.

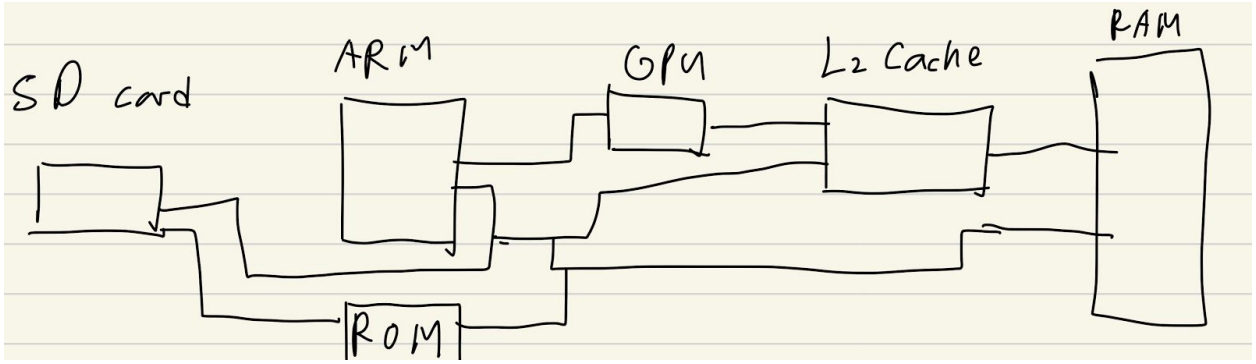
The second function is the file system. So the file system is the system that allows us to have a persistent storage where when the machine is turned off, the files will not disappear. So in Linux we have a directory structure, and also functions and data structures to manage all of the files. For example, in Linux, we have a root directory \ with the subdirectories like home and pi. We also have other subdirectories like temp or etc which are controlled by the administrator.



Another thing that the Linux controls is the dynamic storage. This is how for example in Pi, which has Linux, has dynamic RAM, connected to the processor that runs the kernel and the file system. So when the Pi is powered up, it will perform the boot sequence and will load the kernel into the RAM, and then will run the applications that it sees in a profile (such as the bash rc profile). However, when the power on the Pi is turned off, the dynamic RAM will be blanked out and so it will forget everything stored there. So the Kernel manages the file system and also the dynamic storage on it. It will decide through the virtual memory on who will be using the dynamic storage and also keep it to be blanked out every power off.



5. We have all of these components below which are all interconnected. They are deeply involved in the process of boot sequence in Pi. In the beginning, they are all disabled as no power is supplied.



Then, when power is supplied, the GPU will then be enabled. Then, it will run the Power On Self Test (POST).

Next, it will be the stage 1 of the boot loader. Right after POST, the ROM will be enabled and the initial boot loader is run from the ROM by the GPU. This will enable the SD card and L2 cache and then copies the 32k block from SD card to L2 cache, and executes it. At the end of this step, SD card, ROM, GPU, L2 cache are enabled while RAM and ARM are still disabled.

Next is stage 2, where the Pi will read a file called bootcode.bin from /boot in SD card to L2. This bootcode.bin will enable the RAM and load start.elf (located in /boot) on RAM. So now, only ARM is still disabled. Then, start.efl will enable RAM and reads the config.txt (located in /boot) to take care of the hardware settings and also reads cmdline.txt (located in /boot) to take care of the software settings.

Finally, the kernel.img will be loaded and the control is passed to the init process, which is the process ID one.

The ELF format stands for Executable and Linkable Format. It will have the program header table, which will describe the memory segments that contain information for run-time execution of the file. It will also have the section header table, which will describe the sections, which gives information about linking. The elf files are usually used to boot things up.