List of Operating Systems Projects

Project-1: The concept of the producer-consumer problem was introduced in detailed in the class. So in this project, you need to implement the famous producer consumer problem using Pthreads. The following points should be taken care of properly for successful compilation of the project.

i. Rather than using binary semaphores for empty and full, you will be using standard counting semaphores and mutex lock to replace mutex binary semaphore.
ii. The programs should be thread safe.
iii. Standard mutex locks can be used.

Extend the above module using a multithreaded program that implements the banker's algorithm. The banker will grant the request only if it leaves the system in a safe state. Here also mutex locks will be used to safe data access. A thorough analysis is to be done to showcase the working of the model. The team will get additional credits if they improve the model with different ideas.

Project-2: Standard Linux shell command "apt-get install" allows any Linux user to install packages in the system. During the installation time, several system files are installed and modified. So your job is to perform *hook* (performing additional tasks) both before and after the apt-get install command is executed as suggested below:

i. *Hook before "apt-get install":* The functions that you should perform before hook portion is that you need to check for some specific kernel parameters existing or not in the system (you can take any standard kernel parameters what Linux system has). If the listed parameters exist then further installation will occur else installation procedure will halt. You can find list of kernel parameters at https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt
ii. *Hook after "apt-get install":* On the successful installation (in case there is no halt) many system files will be changed, newly added. So your job is to find all affected files of the system which get modified by this current installation and make a list of them and produce as output.

Project-3: Implement a set of thread-safe functions that update and search an unbalanced binary tree. This library should include functions (with the obvious purposes) of the following form:

- initialize(tree);
- add(tree, char *key, void *value);
- delete(tree, char *key)
- Boolean lookup(char *key, void **value)
- etc.

In the above prototypes, the tree is a structure that points to the root of the tree (you will need to define a suitable structure for this purpose). Each element of the tree holds a key-value pair. You will also need to define the structure for each element to include a mutex that protects that element so that only one thread at a time can access it.

After completing the above task you need to perform the same operation with balanced binary search tree

Then, perform the time and memory comparison between them.

Project-4: The Linux scheduler contains 3 built-in scheduling strategies: SHED_FIFO, SCHED_RR and SCHED_OTHER. The SHED_FIFO and SCHED_RR schedulers are primarily used for real-time scheduling, where a process has time deadlines and requires some guarantee on how soon it will be dispatched. The SCHED_OTHER policy, the default, uses a conventional time-slicing method (similar to round-robin) to put an upper bound on the amount of time that a process will use the CPU. For SCHED_OTHER, a dynamic priority is computed based on a fixed priority and the amount of time a process has been waiting for the CPU to become available. The counter and priority fields in the process control block are the key components in determining the task's dynamic priority. The dynamic priority is adjusted on each timer interrupt.

You are to add a new scheduling policy called SCHED_BACKGROUND that is designed to support processes that only need to run when the system has nothing else to do. This "background" scheduling policy only runs processes when there are no processes in the SCHED_OTHER, SCHED_RR or SCHED_FIFO classes to run. When there is more than one SCHED_BACKGROUND process ready to run, they should compete for the CPU as do SCHED_OTHER processes.

After completing the above steps you need to make a CPU intensive process and perform this process with each of the available scheduling policies to make a time analysis table.

Project-5: The Linux scheduler contains 3 built-in scheduling strategies: SHED_FIFO, SCHED_RR and SCHED_OTHER. The SHED_FIFO and SCHED_RR schedulers are primarily used for real-time scheduling, where a process has time deadlines and requires some guarantee on how soon it will be dispatched. The SCHED_OTHER policy, the default, uses a conventional time-slicing method (similar to round-robin) to put an upper bound on the amount of time that a process will use the CPU. For SCHED_OTHER, a dynamic priority is computed based on a fixed priority and the amount of time a process has been waiting for the CPU to become available. The counter and priority fields in the process control block are the key components in determining the task's dynamic priority. The dynamic priority is adjusted on each timer interrupt.

You are to add a new scheduling policy called SCHED_LOT that is designed to support processes that only need to run when the system has nothing else to do. This "lottery" scheduling policy only runs processes when there are no processes in the SCHED_OTHER, SCHED_RR or SCHED_FIFO classes to run. When there is more than one SCHED_BACKGROUND process ready to run, they should compete for the CPU as do SCHED_OTHER processes.

After completing the above steps you need to make a CPU intensive process and perform this process with each of the available scheduling policies to make a time analysis table.

Project-6: Chose any open source Linux operating system. Consider the input/output redirection operator like pipe (> or <). Write a mini Linux shell having features apart from running the basic commands such as pipe with the help of redirection operators to introduce some controls (that does not exist currently) and remember the history of the commands. Any new justifiable ideas will have better weight.

Project-7: Chose any open source Linux operating system and make some changes to its user interface (like GUI) such that

- Add animations that will be floating dynamically or periodically on the desktop (For example, you can consider IITR events like Cognizance, Thomso, ACM chapter, SDS Labs etc.)
- Sounds while booting or downloading
- Display pop-ups for various operations
- Show a different kind of desktop as per your interest

Project-8: Chose any open source Linux operating system and make some changes to its user interface (like GUI) such that it appears as shown below with the features mentioned working. Like, around 10% of the desktop space for system information, 20% for Apps, 20% for Document Shortcuts (only) (No documents should be allowed to store on desktop), 15% for Latest News (news picked from different sources of favourite news channels opted) and 15% for showcasing to do lists presented from different sources shown in the figure.

Status Bar Day/Date/Time Name of the User Active Apps Other info							Latest News	To Do List
Disks	Apps		Documents*	…				>> Calendar
Disks	Apps		Documents	…			Source 1:	1. 2. …
…	…		…				Source 2:	>> Mail
…			…				…	1. 2. …
…			…				…	>> Other
			Documents				Source 3:	1. 2. …

Project-9: Modify/Improvise any existing device drivers in any type of open source operating systems for any type of architectures, peripherals or components. New ideas will have higher weight.

Project-10: Design (and Implement) any unavailable device drivers for any type of architectures, peripherals or components. If required, simulators may be used for demonstrating your code and results. New ideas will have higher weight.

Project-11: A process "Pause-Restart" mechanism allows a user to pause an existing process, save its state to disk, copy that state to another machine, and restart that process on the new machine. Build a Linux device driver that implements the process "Pause-Restart" mechanism. Be careful to state your assumptions, if any, about the nature of processes that can be paused and restarted using your mechanism.

Project-12: An installation of RedHat Linux takes up several hundred gigabytes (GBs) of disk space. You are supposed to develop a micro-version of Linux that compiles itself and fits on within an MB or two. You should consider removing several functionalities from Linux (such as virtual memory or support for 13 different file systems etc.) as part of this project.