1. Study of Process tree. [programming practice 3.19, 3.20]
2. Design a shell with history feature and signal handling [programming project chapter 3]
3. Add system call to Linux kernel [hint: https://medium.com/anubhav-shrimal/adding-a-hello-world-system-call-to-linux-kernel-dad32875872]

4. The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm: n = { n⁄2, if n is even 3 × n + 1, if n is odd The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1. For example, if n = 35, the sequence is 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1 Write a C program using the fork() system call that generates this sequence in the child process. The starting number will be provided from the command line. For example, if 8 is passed as a parameter on the command line, the child process will output 8, 4, 2, 1. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke the wait() call to wait for the child process to complete before exiting the program. Perform necessary error checking to ensure that a positive integer is passed on the command line. [3,21]

5. The Fibonacci sequence is the series of numbers 0, 1, 1, 2, 3, 5, 8, ... Formally, it can be expressed as: fib0 = 0 fib1 = 1 fibn = fibn−1 + fibn−2 Write a multithreaded program that generates the Fibonacci sequence. This program should work as follows: On the command line, the user will enter the number of Fibonacci numbers that the program is to generate. The program will then create a separate thread that will generate the Fibonacci numbers, placing the sequence in data that can be shared by the threads (an array is probably the most convenient data structure). When the thread finishes execution, the parent thread will output the sequence generated by the child thread. Because the parent thread cannot begin outputting the Fibonacci sequence until the child thread finishes, the parent thread will have to wait for the child thread to finish. [4.27]

6. Write a multi-threading program to computer matrix multiplication of two matrixes. Details in the class.

7. Write a program to simulate the scheduling algorithms

8. Implement Producer-consumer and dining philosopher's problem

9. **Banker's Algorithm** For this project, you will write a program that implements the banker's algorithm discussed in Section 8.6.3. Customers request and release resources from the bank. The banker will grant a request only if it leaves the system in a safe state. A request that leaves the system in an unsafe state will be denied. Although the code examples that describe this project are illustrated in C, you may also develop a solution using Java. The Banker The banker will consider requests from n customers for m resources types, as outlined in Section 8.6.3. The banker will keep track of the resources using the following data structures: #define NUMBER OF CUSTOMERS 5 #define NUMBER OF RESOURCES 4 /* the available amount of each resource */ int available[NUMBER OF RESOURCES]; /*the maximum demand of each customer */ int maximum[NUMBER OF CUSTOMERS][NUMBER OF RESOURCES]; /* the amount currently allocated to each customer */ int allocation[NUMBER OF

CUSTOMERS][NUMBER OF RESOURCES]; /* the remaining need of each customer */ int need[NUMBER OF CUSTOMERS][NUMBER OF RESOURCES]; P-45 Programming Projects The banker will grant a request if it satisfies the safety algorithm outlined in Section 8.6.3.1. If a request does not leave the system in a safe state, the banker will deny it. Function prototypes for requesting and releasing resources are as follows:

int request resources(int customer num, int request[]);

void release resources(int customer num, int release[]);

The request resources() function should return 0 if successful and –1 if unsuccessful. Testing Your Implementation Design a program that allows the user to interactively enter a request for resources, to release resources, or to output the values of the different data structures (available, maximum, allocation, and need) used with the banker's algorithm. You should invoke your program by passing the number of resources of each type on the command line. For example, if there were four resource types, with ten instances of the first type, five of the second type, seven of the third type, and eight of the fourth type, you would invoke your program as follows: ./a.out 10 5 7 8 The available array would be initialized to these values. Your program will initially read in a file containing the maximum number of requests for each customer. For example, if there are five customers and four resources, the input file would appear as follows: 6,4,7,3 4,2,3,2 2,5,3,3 6,3,3,2 5,6,7,5 where each line in the input file represents the maximum request of each resource type for each customer. Your program will initialize the maximum array to these values. Your program will then have the user enter commands responding to a request of resources, a release of resources, or the current values of the different data structures. Use the command 'RQ' for requesting resources, 'RL' for releasing resources, and '*' to output the values of the different data structures. For example, if customer 0 were to request the resources (3, 1, 2, 1), the following command would be entered: RQ 0 3 1 2 1 Your program would then output whether the request would be satisfied or denied using the safety algorithm outlined in text book. Similarly, if customer 4 were to release the resources (1, 2, 3, 1), the user would enter the following command: RL 4 1 2 3 1 Finally, if the command '*' is entered, your program would output the values of the available, maximum, allocation, and need arrays

10. Contiguous Memory Allocation In Section 9.2, we presented different algorithms for contiguous memory allocation. This project will involve managing a contiguous region of memory of size MAX where addresses may range from 0 ... MAX − 1. Your program must respond to four different requests: 1. Request for a contiguous block of memory 2. Release of a contiguous block of memory 3. Compact unused holes of memory into one single block 4. Report the regions of free and allocated memory Your program will be passed the initial amount of memory at startup. For example, the following initializes the program with 1 MB (1,048,576 bytes) of memory: ./allocator 1048576 Once your program has started, it will present the user with the following prompt: allocator> It will then respond to the following commands: RQ (request), RL (release), C (compact), STAT (status report), and X (exit). A request for 40,000 bytes will appear as follows: allocator>RQ P0 40000 W P-48 Chapter 9 Main Memory Similarly, a release will appear as: allocator>RL P0 This command will release the memory that has been allocated to process P0. The command for compaction is entered as: allocator>C This command will compact unused holes of memory into one region. Finally, the STAT command for reporting the status of memory is entered as: allocator>STAT Given this command, your program will report the regions of memory that are allocated and the regions that are unused. For example, one possible arrangement of memory allocation would be as follows: Addresses [0:315000] Process P1 Addresses [315001: 512500] Process P3 Addresses

[512501:625575] Unused Addresses [625575:725100] Process P6 Addresses [725001] . . . Allocating Memory Your program will allocate memory using one of the three approaches highlighted in Section 9.2.2, depending on the flag that is passed to the RQ command. The flags are: • F—first fit • B—best fit • W—worst fit This will require that your program keep track of the different holes representing available memory. When a request for memory arrives, it will allocate the memory from one of the available holes based on the allocation strategy. If there is insufficient memory to allocate to a request, it will output an error message and reject the request. Your program will also need to keep track of which region of memory has been allocated to which process. This is necessary to support the STAT The first parameter to the RQ command is the new process that requires the memory, followed by the amount of memory being requested, and finally the strategy. (In this situation, “W” refers to worst fit.) command and is also needed when memory is released via the RL command, as the process releasing memory is passed to this command. If a partition being released is adjacent to an existing hole, be sure to combine the two holes into a single hole