Operating Systems Homework 3 Report

Emre Çamlıca - 150210071,

1 Introduction

In this homework, I designed a safety check for the Banker's algorithm, using file operations.

2 Design

In my design I first created 4 arrays, three of them to keep the values read from the file and one to store the processes that have finished their requests.1 shows an example of file read operation I did for storing the resources file into the resources array. I used a while loop so that the program will read the file until reaching the end. I stored the values read one by one to the resources array using fscanf function. The code for allocations and requests arrays look similar, however, I used double for loops to store their elements as they are two dimensional arrays.

```
while (!feof(fptr))
{
    for(i=0; i<5; i++){
        fscanf(fptr, "%d", &val);
        resources[i] = val;
    }
}
fclose(fptr);
```

Figure 1: Example file read operation

After reading the file, I subtracted the allocated resources from the all resources to get the currently available resources, using a double for loop to traverse the allocations array.

In the safety check code, I used three functions: find available, allocate and release:

The find available function takes resources, requests and finished arrays as parameters, searches the non finished processes of the requests array, and if it finds an available process that can allocate sufficient resources depending on its requests, it returns its index. If not, it returns -1.

The allocate function takes the four arrays and an integer n as a parameter. It sets the finished index corresponding to the process as 1, which is checked in the find available function. After that, in a for loop, it takes the requested resources from the resources array, increases the corresponding resources in the allocations array and sets the requests of the process as zero.

The release function releases all the resources allocated by the process, then sets all the allocations of it as zero, in a double for loop.

```
printf("\nRunning order for processes: ");
while(finished[0]+finished[1]+finished[2]+finished[3]+finished[4]!=5){
n=find_available(resources, requests, finished);

if(n==-1){
    printf("\nThere is a deadlock. ");
    for(int i=0; i<5; i++){
        if(finished[i]==0){
            printf("P%d ", i+1);
        }
    }
    printf("are the cause of deadlock.");
    break;
}
allocate(n, resources, requests, allocations, finished);
printf("P%d ", n+1);
while (n!=-1)
{
    n=find_available(resources, requests, finished);
    if(n==-1)
        break;

    allocate(n, resources, requests, allocations, finished);
    printf("P%d ", n+1);
}
release(resources, allocations, finished);
}
```

Figure 2: Safety Check Code

1 shows the safety check code for Banker's algorithm. The while loop condition is satisfied when there is no possibility of a deadlock for the inputs read, where all processes are finished successfully. In the algorithm, I used the integer n to check if there are currently available processes, if so, to store their indexes. First, it tries to find an available process. If there aren't any available processes and all processes are not finished yet, this means there is the possibility of a deadlock. If there is an available process it is allocated. Since the function checks the worst case, it searches for other available processes while one process is still working. At the end of the while loop, all processes are released.