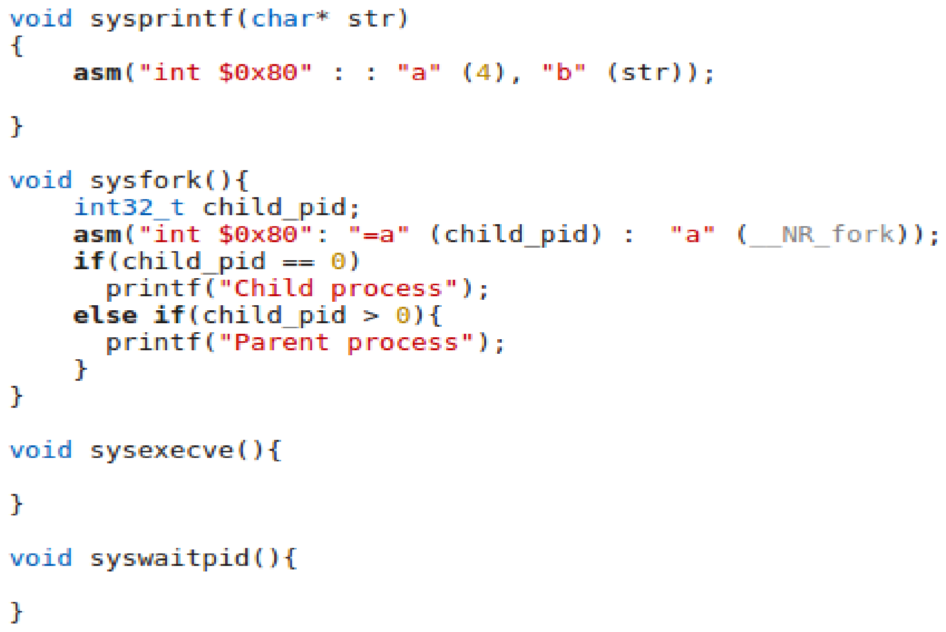
CSE 312 HOMEWORK 1 REPORT

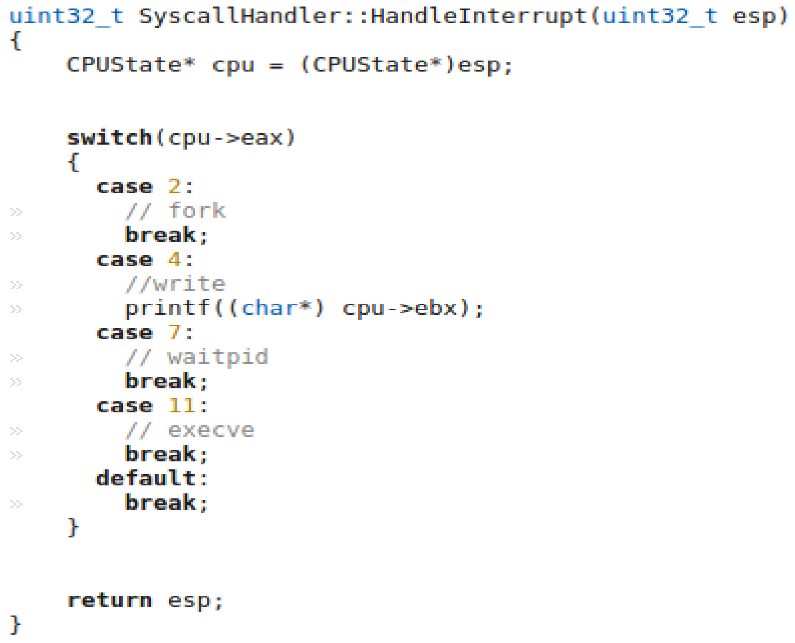
Creating System Calls



```
void sysprintf(char* str)
{
    asm("int $0x80" : : "a" (4), "b" (str));

}

void sysfork(){
    int32_t child_pid;
    asm("int $0x80": "=a" (child_pid) :  "a" (__NR_fork));
    if(child_pid == 0)
      printf("Child process");
    else if(child_pid > 0){
      printf("Parent process");
    }
}

void sysexecve(){

}

void syswaitpid(){

}
```

Some of the system calls are implemented but execve and waitpid cannot be implemented because the videos are not clear. There was no example about how to implement system calls.

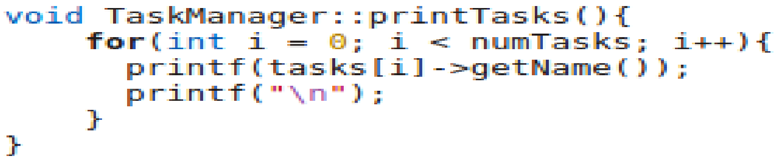
Handling System Calls



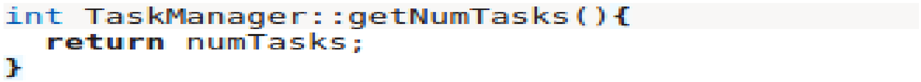
```
uint32_t SyscallHandler::HandleInterrupt(uint32_t esp)
{
    CPUState* cpu = (CPUState*)esp;


    switch(cpu->eax)
    {
      case 2:
        // fork
        break;
      case 4:
        //write
        printf((char*) cpu->ebx);
      case 7:
        // waitpid
        break;
      case 11:
        // execve
        break;
      default:
        break;
    }


    return esp;
}
```

The reason why handling system calls is not implemented properly is that videos are not well-explained and there is not enough resource.

Displaying Tasks

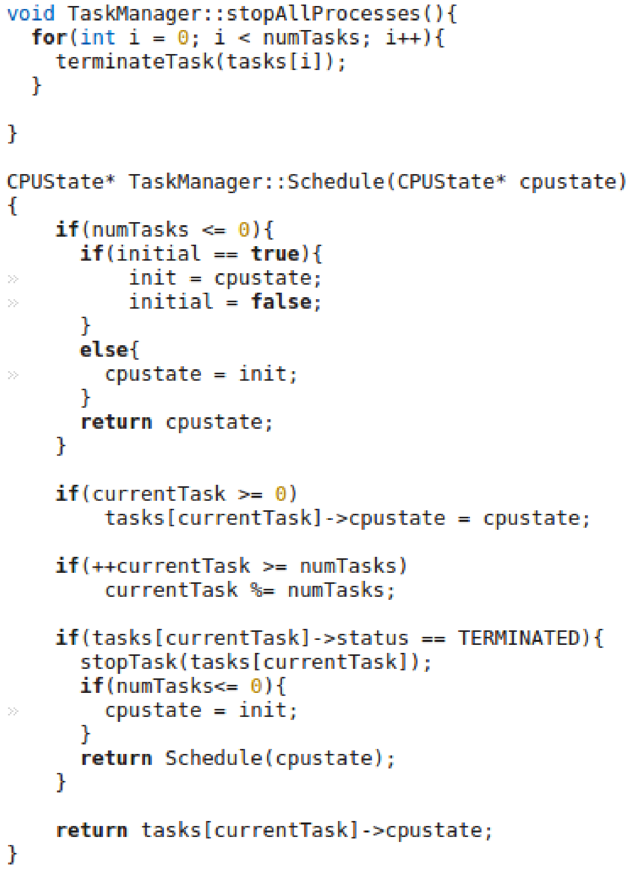


```
void TaskManager::printTasks(){
    for(int i = 0; i < numTasks; i++){
      printf(tasks[i]->getName());
      printf("\n");
    }
}
```



```
int TaskManager::getNumTasks(){
  return numTasks;
}
```

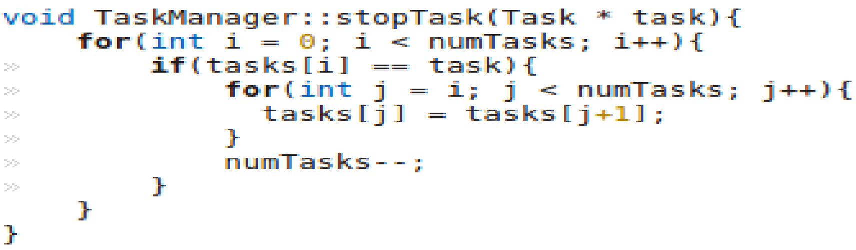
Task Manager Functionalities



```
void TaskManager::stopAllProcesses(){
  for(int i = 0; i < numTasks; i++){
    terminateTask(tasks[i]);
  }

}

CPUState* TaskManager::Schedule(CPUState* cpustate)
{
    if(numTasks <= 0){
      if(initial == true){
          init = cpustate;
          initial = false;
      }
      else{
        cpustate = init;
      }
      return cpustate;
    }

    if(currentTask >= 0)
        tasks[currentTask]->cpustate = cpustate;

    if(++currentTask >= numTasks)
        currentTask %= numTasks;

    if(tasks[currentTask]->status == TERMINATED){
      stopTask(tasks[currentTask]);
      if(numTasks<= 0){
        cpustate = init;
      }
      return Schedule(cpustate);
    }

    return tasks[currentTask]->cpustate;
}
```

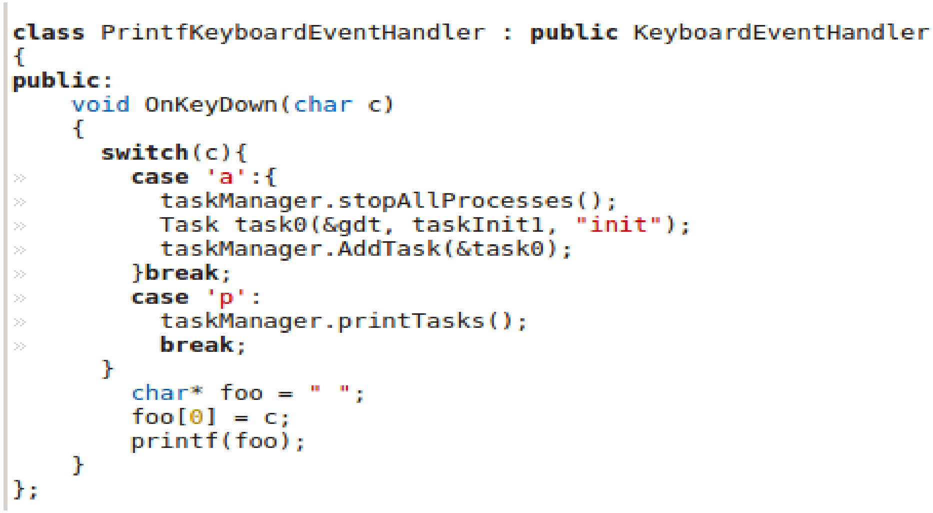
stopAllProcesses function terminates all processes in the cpu.

In currentTask %= numTasks line in Schedule function, Round Robin Scheduling is performed. The next if statement after the Round Robin Scheduling is provided to remove terminated tasks from the task manager.



```
void TaskManager::stopTask(Task * task){
    for(int i = 0; i < numTasks; i++){
        if(tasks[i] == task){
            for(int j = i; j < numTasks; j++){
              tasks[j] = tasks[j+1];
            }
            numTasks--;
        }
    }
}
```

User Interactions

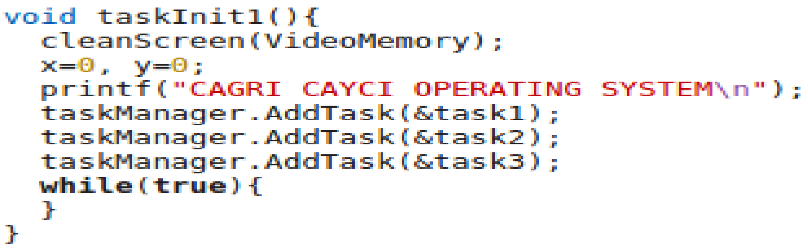


```
class PrintfKeyboardEventHandler : public KeyboardEventHandler
{
public:
    void OnKeyDown(char c)
    {
      switch(c){
        case 'a':{
          taskManager.stopAllProcesses();
          Task task0(&gdt, taskInit1, "init");
          taskManager.AddTask(&task0);
        }break;
        case 'p':
          taskManager.printTasks();
          break;
      }
        char* foo = " ";
        foo[0] = c;
        printf(foo);
    }
};
```

If a key is pressed, all processes on the task manager are terminated and init process is added.

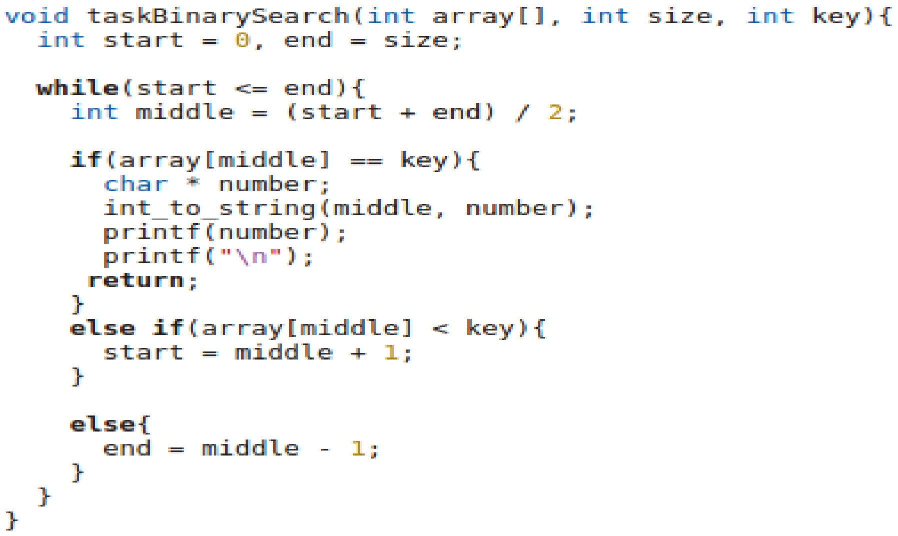
If p key is pressed, all processes on the task manager are printed.

Tasks



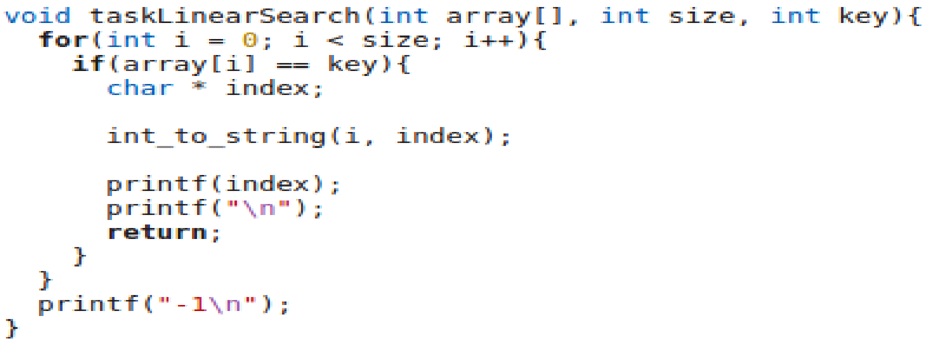
```
void taskInit1(){
  cleanScreen(VideoMemory);
  x=0, y=0;
  printf("CAGRI CAYCI OPERATING SYSTEM\n");
  taskManager.AddTask(&task1);
  taskManager.AddTask(&task2);
  taskManager.AddTask(&task3);
  while(true){
  }
}
```

Init task is loaded binary search task, collatz task and linear search task in the memory and added them to task manager.



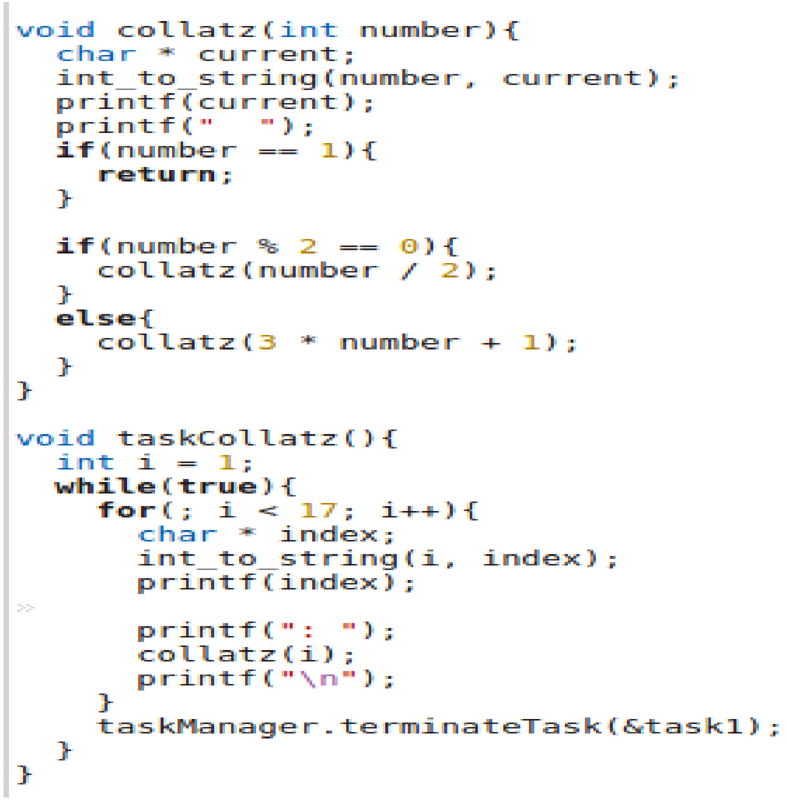
```
void taskBinarySearch(int array[], int size, int key){
  int start = 0, end = size;

  while(start <= end){
    int middle = (start + end) / 2;

    if(array[middle] == key){
      char * number;
      int_to_string(middle, number);
      printf(number);
      printf("\n");
     return;
    }
    else if(array[middle] < key){
      start = middle + 1;
    }

    else{
      end = middle - 1;
    }
  }
}
```

taskBinarySearch function search the key in the sorted array.



```
void taskLinearSearch(int array[], int size, int key){
  for(int i = 0; i < size; i++){
    if(array[i] == key){
      char * index;

      int_to_string(i, index);

      printf(index);
      printf("\n");
      return;
    }
  }
  printf("-1\n");
}
```

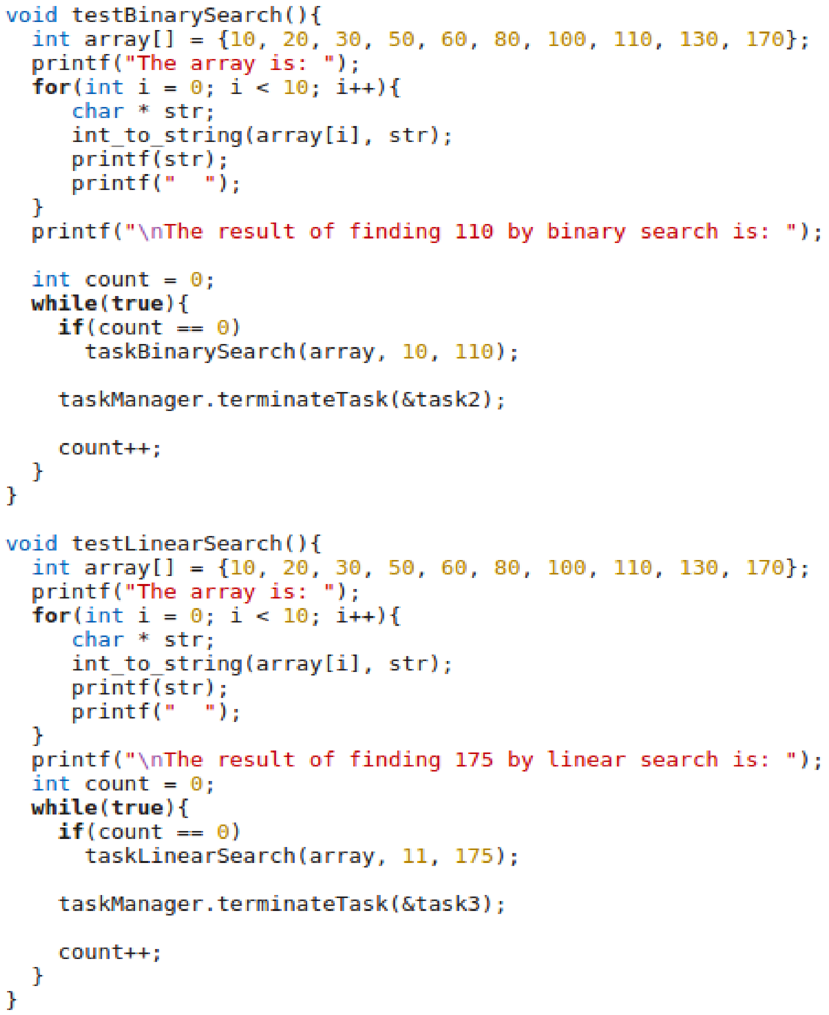
taskLinearSearch function search the key in the array.



```
void collatz(int number){
   char * current;
   int_to_string(number, current);
   printf(current);
   printf("   ");
   if(number == 1){
      return;
   }

   if(number % 2 == 0){
      collatz(number / 2);
   }
   else{
      collatz(3 * number + 1);
   }
}

void taskCollatz(){
   int i = 1;
   while(true){
      for(; i < 17; i++){
         char * index;
         int_to_string(i, index);
         printf(index);

         printf(": ");
         collatz(i);
         printf("\n");
      }
      taskManager.terminateTask(&task1);
   }
}
```

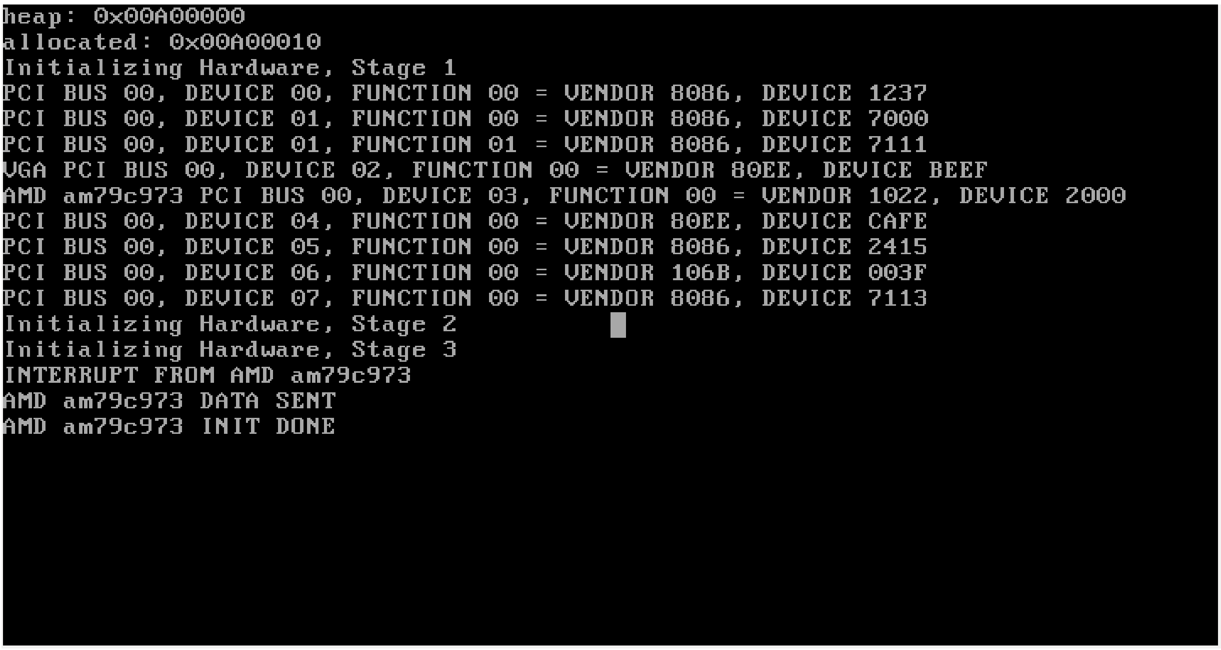
collatz function does the task which is defined in the homework pdf.

taskCollatz calls collatz function 17 times and terminates the task.

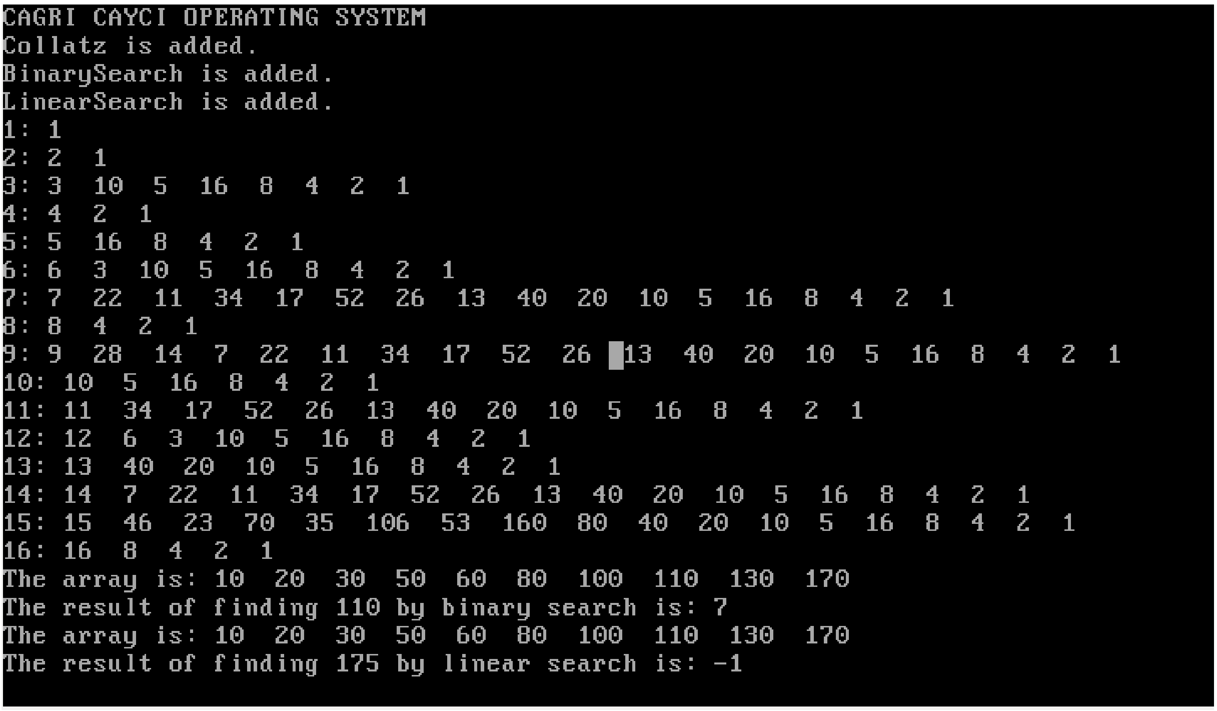


```
void testBinarySearch(){
  int array[] = {10, 20, 30, 50, 60, 80, 100, 110, 130, 170};
  printf("The array is: ");
  for(int i = 0; i < 10; i++){
     char * str;
     int_to_string(array[i], str);
     printf(str);
     printf("  ");
  }
  printf("\nThe result of finding 110 by binary search is: ");

  int count = 0;
  while(true){
    if(count == 0)
      taskBinarySearch(array, 10, 110);

    taskManager.terminateTask(&task2);

    count++;
  }
}

void testLinearSearch(){
  int array[] = {10, 20, 30, 50, 60, 80, 100, 110, 130, 170};
  printf("The array is: ");
  for(int i = 0; i < 10; i++){
     char * str;
     int_to_string(array[i], str);
     printf(str);
     printf("  ");
  }
  printf("\nThe result of finding 175 by linear search is: ");
  int count = 0;
  while(true){
    if(count == 0)
      taskLinearSearch(array, 11, 175);

    taskManager.terminateTask(&task3);

    count++;
  }
}
```

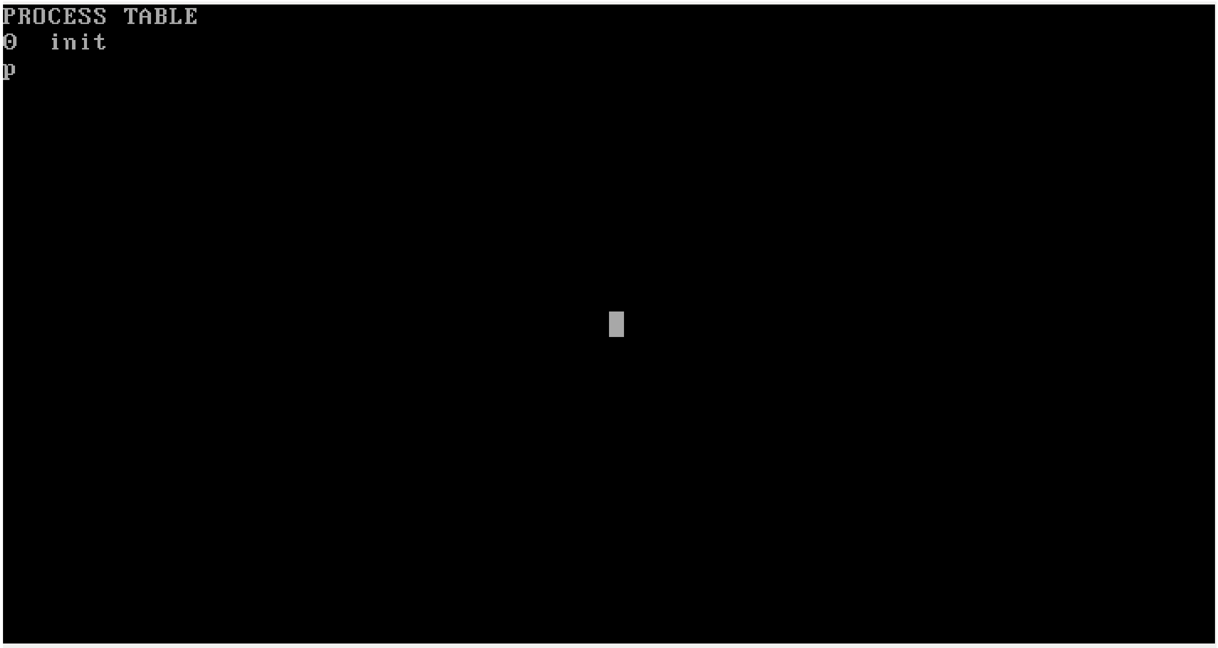
Tests



```
heap: 0x00A00000
allocated: 0x00A00010
Initializing Hardware, Stage 1
PCI BUS 00, DEVICE 00, FUNCTION 00 = VENDOR 8086, DEVICE 1237
PCI BUS 00, DEVICE 01, FUNCTION 00 = VENDOR 8086, DEVICE 7000
PCI BUS 00, DEVICE 01, FUNCTION 01 = VENDOR 8086, DEVICE 7111
VGA PCI BUS 00, DEVICE 02, FUNCTION 00 = VENDOR 80EE, DEVICE BEEF
AMD am79c973 PCI BUS 00, DEVICE 03, FUNCTION 00 = VENDOR 1022, DEVICE 2000
PCI BUS 00, DEVICE 04, FUNCTION 00 = VENDOR 80EE, DEVICE CAFE
PCI BUS 00, DEVICE 05, FUNCTION 00 = VENDOR 8086, DEVICE 2415
PCI BUS 00, DEVICE 06, FUNCTION 00 = VENDOR 106B, DEVICE 003F
PCI BUS 00, DEVICE 07, FUNCTION 00 = VENDOR 8086, DEVICE 7113
Initializing Hardware, Stage 2
Initializing Hardware, Stage 3
INTERRUPT FROM AMD am79c973
AMD am79c973 DATA SENT
AMD am79c973 INIT DONE
```



```
CAGRI CAYCI OPERATING SYSTEM
Collatz is added.
BinarySearch is added.
LinearSearch is added.
1: 1
2: 2  1
3: 3  10  5  16  8  4  2  1
4: 4  2  1
5: 5  16  8  4  2  1
6: 6  3  10  5  16  8  4  2  1
7: 7  22  11  34  17  52  26  13  40  20  10  5  16  8  4  2  1
8: 8  4  2  1
9: 9  28  14  7  22  11  34  17  52  26  13  40  20  10  5  16  8  4  2  1
10: 10  5  16  8  4  2  1
11: 11  34  17  52  26  13  40  20  10  5  16  8  4  2  1
12: 12  6  3  10  5  16  8  4  2  1
13: 13  40  20  10  5  16  8  4  2  1
14: 14  7  22  11  34  17  52  26  13  40  20  10  5  16  8  4  2  1
15: 15  46  23  70  35  106  53  160  80  40  20  10  5  16  8  4  2  1
16: 16  8  4  2  1
The array is: 10  20  30  50  60  80  100  110  130  170
The result of finding 110 by binary search is: 7
The array is: 10  20  30  50  60  80  100  110  130  170
The result of finding 175 by linear search is: -1
```



```
PROCESS TABLE
0  init
p
```