Indian Institute of Technology, Delhi

ELL 783 – Operating Systems

Assignment 1 - Easy

Name: Ranjan kumar Singha

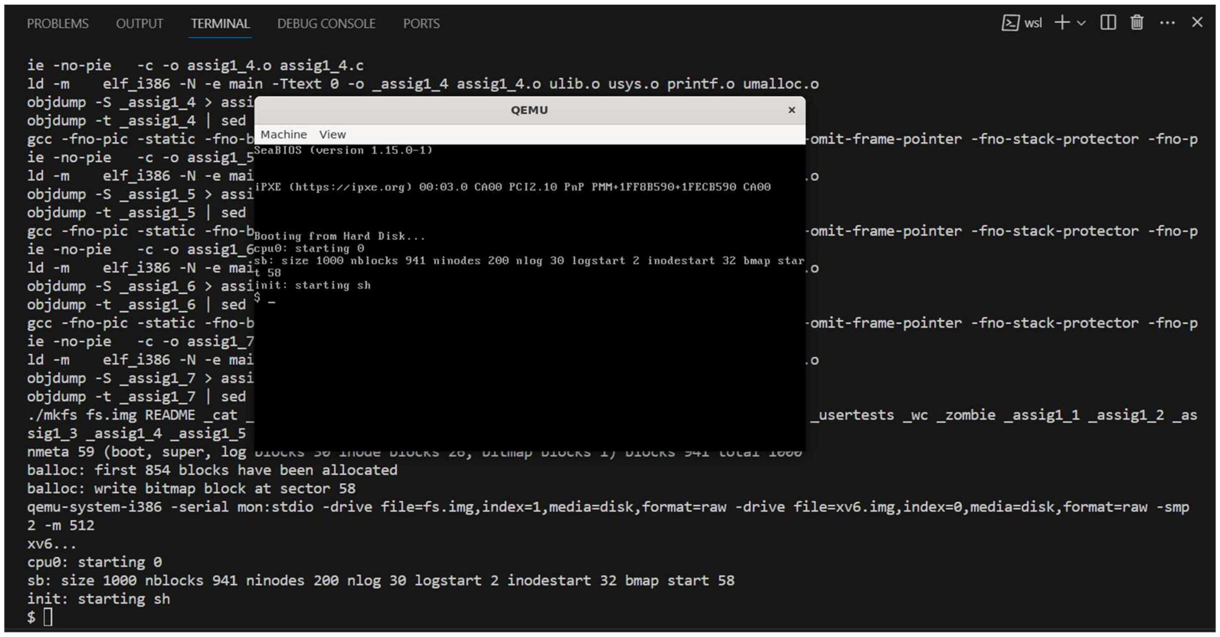
Entry No.: 2023EET2192

Course: M.Tech. in Computer Technology

Department: Electrical Engineering

Supervisor: Prof. Smruti R. Sarangi

1. Installing and Testing Xv6:



```
ie -no-pie   -c -o assig1_4.o assig1_4.c
ld -m    elf_i386 -N -e main -Ttext 0 -o _assig1_4 assig1_4.o ulib.o usys.o printf.o umalloc.o
objdump -S _assig1_4 > assi
objdump -t _assig1_4 | sed
gcc -fno-pic -static -fno-b                                   -omit-frame-pointer -fno-stack-protector -fno-p
ie -no-pie   -c -o assig1_5
ld -m    elf_i386 -N -e mai                                   .o
objdump -S _assig1_5 > assi
objdump -t _assig1_5 | sed
gcc -fno-pic -static -fno-b                                   -omit-frame-pointer -fno-stack-protector -fno-p
ie -no-pie   -c -o assig1_6
ld -m    elf_i386 -N -e mai                                   .o
objdump -S _assig1_6 > assi
objdump -t _assig1_6 | sed
gcc -fno-pic -static -fno-b                                   -omit-frame-pointer -fno-stack-protector -fno-p
ie -no-pie   -c -o assig1_7
ld -m    elf_i386 -N -e mai                                   .o
objdump -S _assig1_7 > assi
objdump -t _assig1_7 | sed
./mkfs fs.img README _cat _                                   _usertests _wc _zombie _assig1_1 _assig1_2 _as
sig1_3 _assig1_4 _assig1_5
nmeta 59 (boot, super, log blocks 30 inode blocks 26, bitmap blocks 1) blocks 941 total 1000
balloc: first 854 blocks have been allocated
balloc: write bitmap block at sector 58
qemu-system-i386 -serial mon:stdio -drive file=fs.img,index=1,media=disk,format=raw -drive file=xv6.img,index=0,media=disk,format=raw -smp
2 -m 512
xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ 
```

```
SeaBIOS (version 1.15.0-1)

iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B590+1FECB590 CA00

Booting from Hard Disk...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap star
t 58
init: starting sh
$ _
```

2. System Calls:

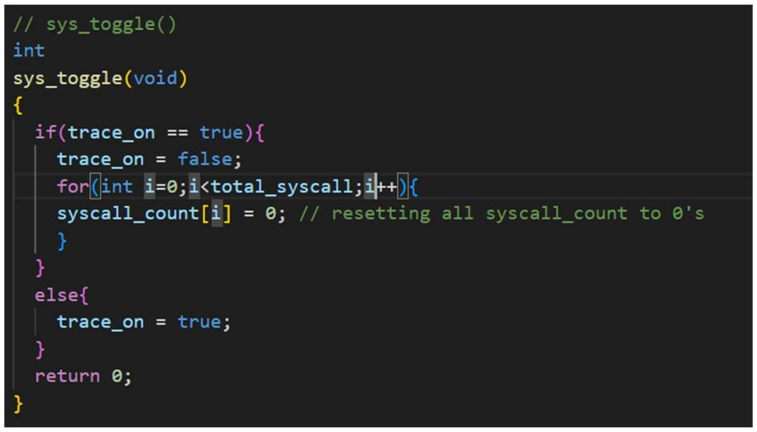
To add a system call in Xv6, we have to make changes in some following files of Xv6:

- ***syscall.h*** *: We need to set a unique number to each system call here.*
- ***syscall.c*** *: We need to add a pointer to the system call function here.*
- ***sysproc.c*** *: We need to define the system call function here.*
- ***usys.s*** *: : We need to add an entry keeping with the convention so that assembly stub can move the system call number into reg eax and make the system call.*

- ***user.h** : We need to add an entry so that system call is visible to the user.*

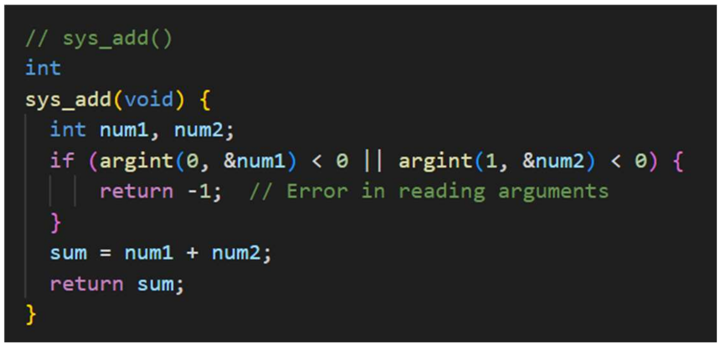
A. ***sys_print_count():*** *In syscall.c I defined a variable called "trace_on" assigned with false initially, and an array named "system_call_count[29]" (29 because we have 28 total number of system calls in this assignment, and 0 is not numbered to any system call) assigned all indices with value 0 initially. Now in syscall() function I check if trace_on is true, if so then I increment the system_call_count[] array's particular index that is assigned with corresponding system call. Then I came to sysproc.c file, where I've implemented the sys_print_count() system call, where I used if conditions sequentially according to the alphabetical order of the system call names and print accordingly.*

B. ***sys_toggle():*** *In syscall.c I defined a variable called "trace_on" assigned with true initially, then I came to sysproc.c file where I've implemented sys_toggle() system call, where if trace_on was false, then I made it true; else if trace_on was true, I made it false and reset all the system call count to 0.*



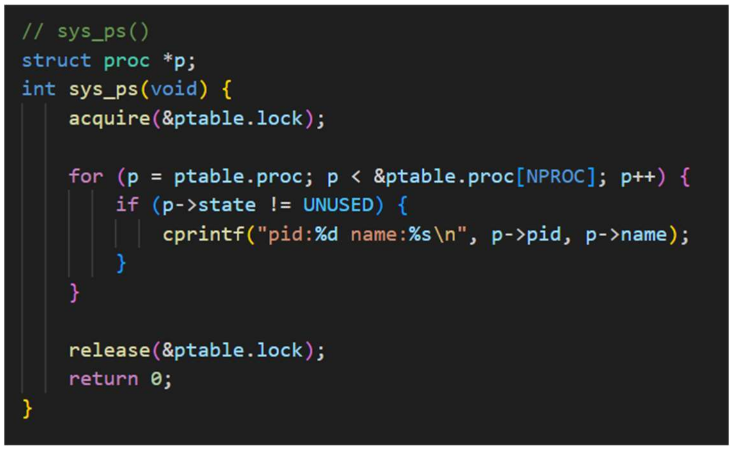
```
// sys_toggle()
int
sys_toggle(void)
{
  if(trace_on == true){
    trace_on = false;
    for(int i=0;i<total_syscall;i++){
    syscall_count[i] = 0; // resetting all syscall_count to 0's
    }
  }
  else{
    trace_on = true;
  }
  return 0;
}
```

C. ***sys_add():*** *In sysproc.c file, I've implemented the sys_add() system call, where it takes two integer arguments, and then return the sum after doing the calculation.*



```
// sys_add()
int
sys_add(void) {
  int num1, num2;
  if (argint(0, &num1) < 0 || argint(1, &num2) < 0) {
      return -1;  // Error in reading arguments
  }
  sum = num1 + num2;
  return sum;
}
```

D. ***sys_ps():*** *In sysproc.c file, I've implemented the sys_ps() system call, where it traverse the whole process table and prints the process id and process name that is running or runnable.*



```
// sys_ps()
struct proc *p;
int sys_ps(void) {
    acquire(&ptable.lock);

    for (p = ptable.proc; p < &ptable.proc[NPROC]; p++) {
        if (p->state != UNUSED) {
            cprintf("pid:%d name:%s\n", p->pid, p->name);
        }
    }

    release(&ptable.lock);
    return 0;
}
```

3. Inter – process Communication:

UNICAST MODEL

Interprocess communication (IPC) is a fundamental concept in operating systems, allowing processes to exchange data and synchronize their activities. Unicast IPC involves communication between two specific processes, where one process sends a message to another process.

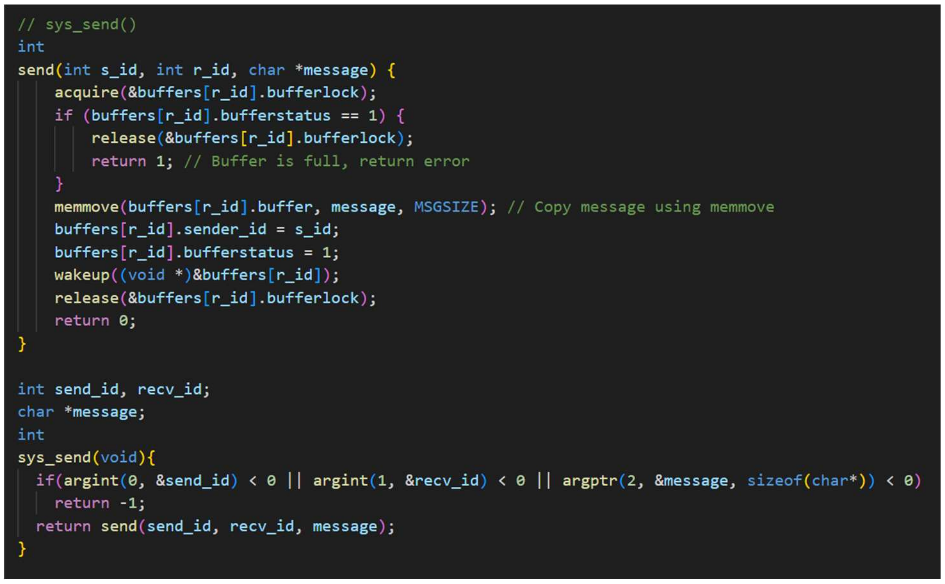
Implementation Overview: In this report, we present the implementation of unicast IPC in an operating system environment. Our implementation consists of two main functions: sys_send() and sys_recv(), along with supporting data structures and synchronization mechanisms.

Data Structures:

- *Buffer Structure: We define a buffer_struct to hold the message, sender ID, buffer status, and a spinlock for synchronization. Each buffer is associated with a specific receiver process.*
- *Buffer Array: An array of buffer_struct objects is used to manage communication between multiple sender and receiver pairs.*

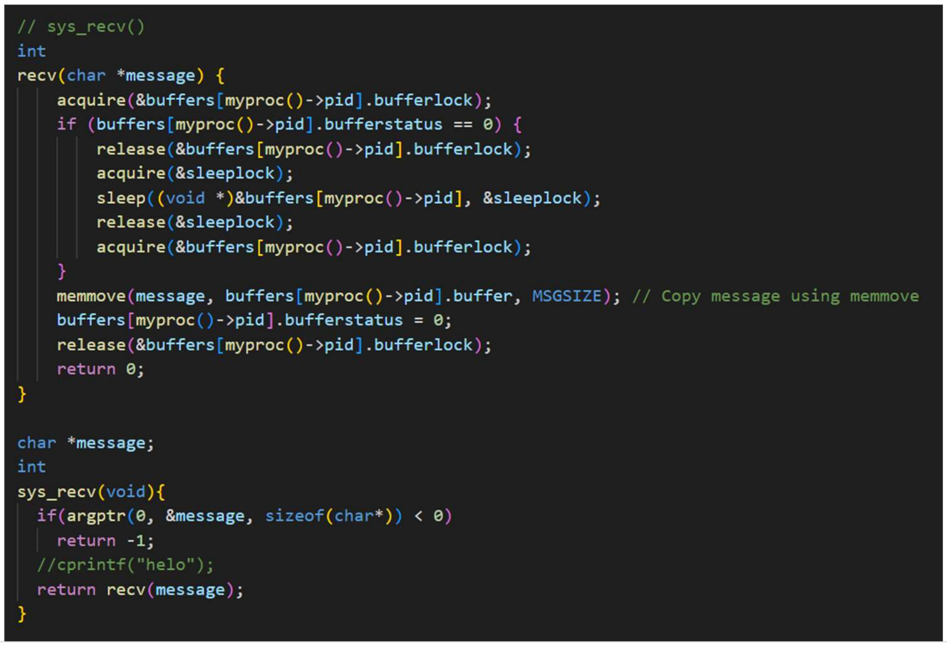
Functions:

- **send(int send_id, int recv_id, char message): This function is responsible for sending a message from a sender process (send_id) to a receiver process (recv_id). It acquires the buffer lock associated with the receiver, checks if the buffer is available, copies the message into the buffer, updates the sender ID and buffer status, and wakes up the receiver process.*
- *sys_send(): This system call function interfaces with the send() function, extracting sender ID, receiver ID, and message pointer from the arguments passed by the user.*
- **recv(char message): This function is responsible for receiving a message in the current process. It acquires the buffer lock associated with the current process, checks if the buffer contains a message, copies the message from the buffer, updates the buffer status, and releases the buffer lock.*
- *sys_recv(): This system call function interfaces with the recv() function, extracting the message pointer from the argument passed by the user.*



```
// sys_send()
int
send(int s_id, int r_id, char *message) {
    acquire(&buffers[r_id].bufferlock);
    if (buffers[r_id].bufferstatus == 1) {
        release(&buffers[r_id].bufferlock);
        return 1; // Buffer is full, return error
    }
    memmove(buffers[r_id].buffer, message, MSGSIZE); // Copy message using memmove
    buffers[r_id].sender_id = s_id;
    buffers[r_id].bufferstatus = 1;
    wakeup((void *)&buffers[r_id]);
    release(&buffers[r_id].bufferlock);
    return 0;
}

int send_id, recv_id;
char *message;
int
sys_send(void){
  if(argint(0, &send_id) < 0 || argint(1, &recv_id) < 0 || argptr(2, &message, sizeof(char*)) < 0)
    return -1;
  return send(send_id, recv_id, message);
}
```



```
// sys_recv()
int
recv(char *message) {
    acquire(&buffers[myproc()->pid].bufferlock);
    if (buffers[myproc()->pid].bufferstatus == 0) {
        release(&buffers[myproc()->pid].bufferlock);
        acquire(&sleeplock);
        sleep((void *)&buffers[myproc()->pid], &sleeplock);
        release(&sleeplock);
        acquire(&buffers[myproc()->pid].bufferlock);
    }
    memmove(message, buffers[myproc()->pid].buffer, MSGSIZE); // Copy message using memmove
    buffers[myproc()->pid].bufferstatus = 0;
    release(&buffers[myproc()->pid].bufferlock);
    return 0;
}

char *message;
int
sys_recv(void){
  if(argptr(0, &message, sizeof(char*)) < 0)
    return -1;
  //cprintf("helo");
  return recv(message);
}
```

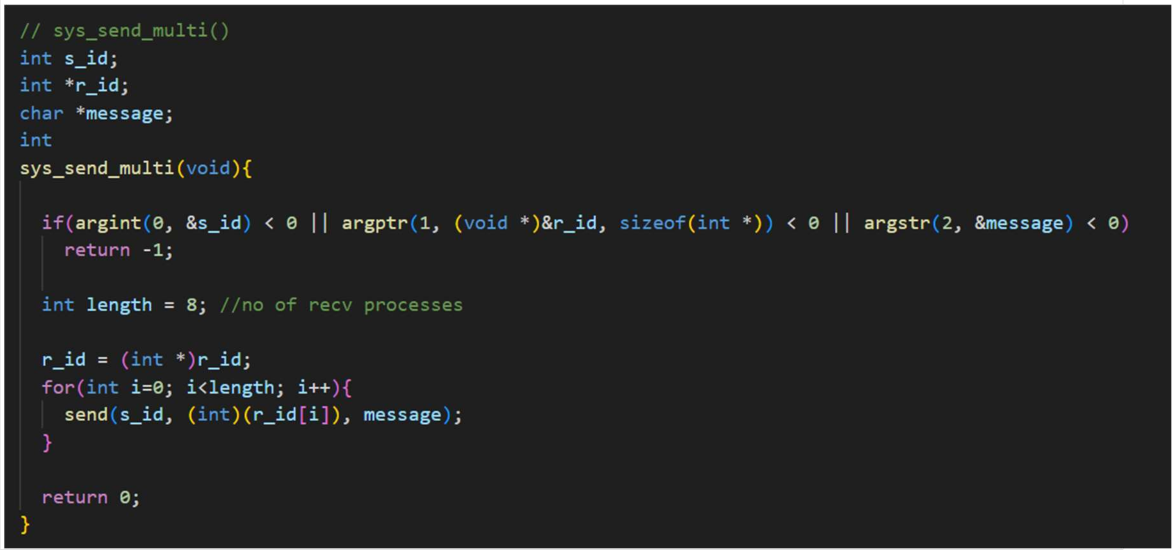
Multicast Model:

In this section, we discuss the extension of the unicast interprocess communication (IPC) model to a multicast model, leveraging the sys_send() and *sys_recv() system calls. Multicast communication enables a single sender process to transmit messages to multiple receiver processes simultaneously, enhancing the flexibility and efficiency of interprocess communication within the operating system.*

*Implementation Details: To implement multicast communication, we introduce a new system call, sys_send_multi(int, int *, void *), which accepts the sender ID, an array of receiver IDs, and the message as arguments. The number of receiver processes is assumed to be a constant, set to 8 in this implementation.*

Algorithm:

1. *Accept the sender ID (s_id), an array of receiver IDs (r_ids), and the message (msg) as input arguments.*
2. *Iterate over each receiver ID in the array r_ids.*
3. *For each receiver ID:*
    - *Invoke the sys_send() system call, passing the sender ID (s_id), receiver ID (r_id), and message (msg).*
    - *The sys_send() call places the message into the buffer of the corresponding receiver process, where it can be accessed via the sys_recv() system call.*
    - *Upon successful transmission, the receiver process is awakened to read the message from its buffer.*



```
// sys_send_multi()
int s_id;
int *r_id;
char *message;
int
sys_send_multi(void){

  if(argint(0, &s_id) < 0 || argptr(1, (void *)&r_id, sizeof(int *)) < 0 || argstr(2, &message) < 0)
    return -1;

  int length = 8; //no of recv processes

  r_id = (int *)r_id;
  for(int i=0; i<length; i++){
    send(s_id, (int)(r_id[i]), message);
  }

  return 0;
}
```

4. ***Distributed Algorithm***:

A. ***Calculating Total Sum***:

In this task, we aim to compute the sum of 1000 elements using a parallel computing approach. A coordinator process divides the task among 8 child processes, each of which computes its partial sum and sends it back to the coordinator. The coordinator then aggregates these partial sums to obtain the total sum.

Implementation Overview: The implementation involves initializing necessary variables, forking child processes, computing partial sums in child processes, and aggregating them in the coordinator process. Key components of the implementation include process management, partial sum computation, and interprocess communication using sys_send() and sys_recv() system calls.

Initialization:

- *noOfProcesses: Number of child processes (8 in this case).*
- *pidParent: PID of the coordinator process.*
- *elem_per_proc: Number of elements each process needs to compute.*
- *partialSumP and pmsg: Buffers for storing partial sums during communication.*
- *partialSum: Variable for computing partial sums.*

Partial Sum Computation:

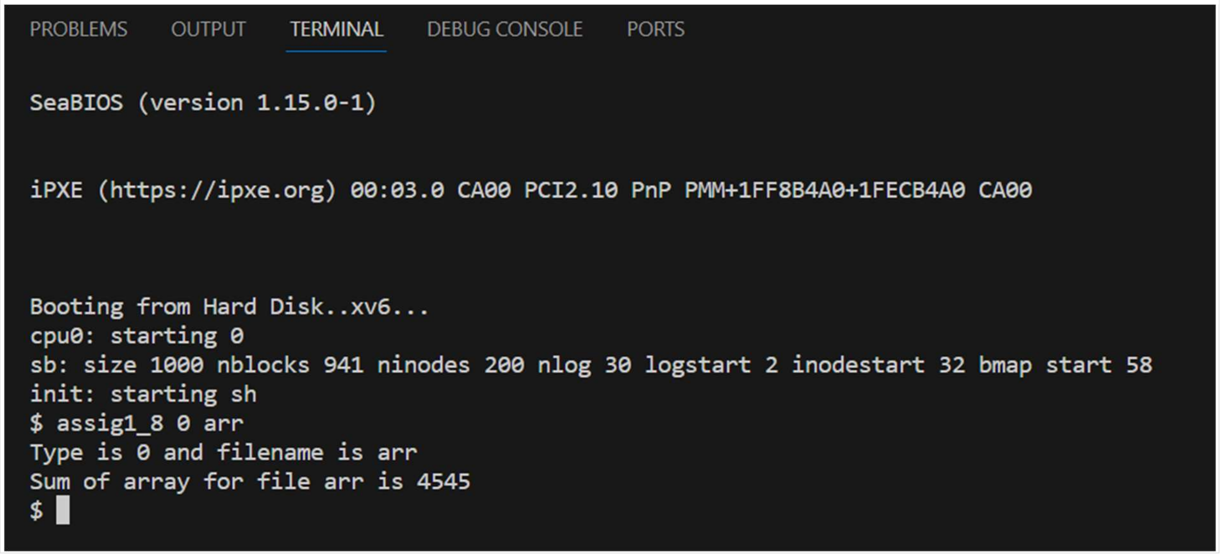
- *A loop iterates over the number of child processes.*
- *Each iteration forks a child process, ensuring each child has a copy of the parent's address space.*
- *Child processes compute their respective partial sums based on assigned array elements.*

- *Partial sums are sent to the coordinator process using sys_send() and sys_recv() calls.*
- *Child processes exit after sending their partial sums.*

Aggregation in Coordinator:

- *The coordinator process waits for each child process to exit using wait().*
- *It receives partial sums from child processes using sys_recv() and aggregates them to obtain the total sum.*

Conclusion: The parallel sum calculation approach efficiently distributes the computational load among multiple processes, leveraging interprocess communication for coordination. By dividing the task and aggregating results, the system achieves parallelism and accelerates computation. This implementation demonstrates the effectiveness of parallel computing in handling computationally intensive tasks.



```
SeaBIOS (version 1.15.0-1)


iPXE (https://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8B4A0+1FECB4A0 CA00



Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
init: starting sh
$ assig1_8 0 arr
Type is 0 and filename is arr
Sum of array for file arr is 4545
$ 
```