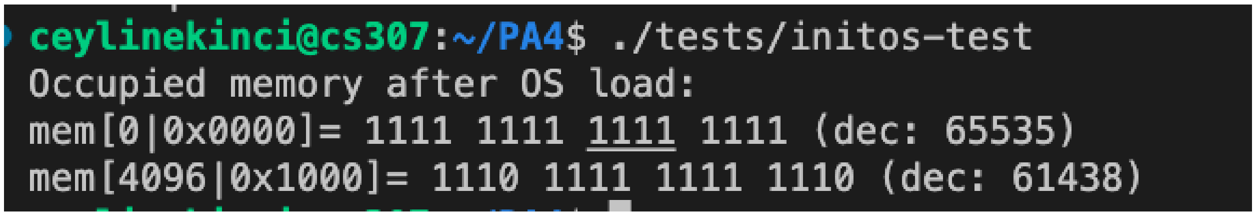
Extending Virtual Memory Implementation with Segmentation
Report
Ceylin Ekinci 28369

In this project, we extended a simple virtual memory implementation by adding segmentation and implementing system calls for handling multiple processes. This report details our implementation of the new requirements, including address translation mechanisms, memory allocation, and process management.

1. Initializing the OS (initOS)

The initOS function initializes the OS-related parts of the physical memory, setting up necessary metadata and the free list.

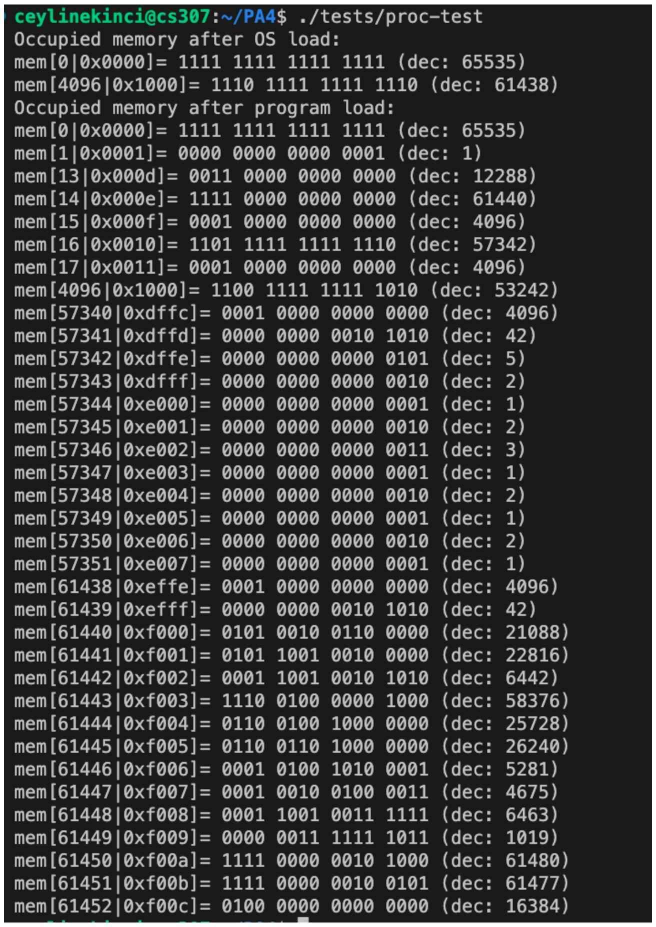
Initos-test works correctly.



```
ceylinekinci@cs307:~/PA4$ ./tests/initos-test
Occupied memory after OS load:
mem[0|0x0000]= 1111 1111 1111 1111 (dec: 65535)
mem[4096|0x1000]= 1110 1111 1111 1110 (dec: 61438)
```

2. Creating and Loading Processes (createProc and loadProc)

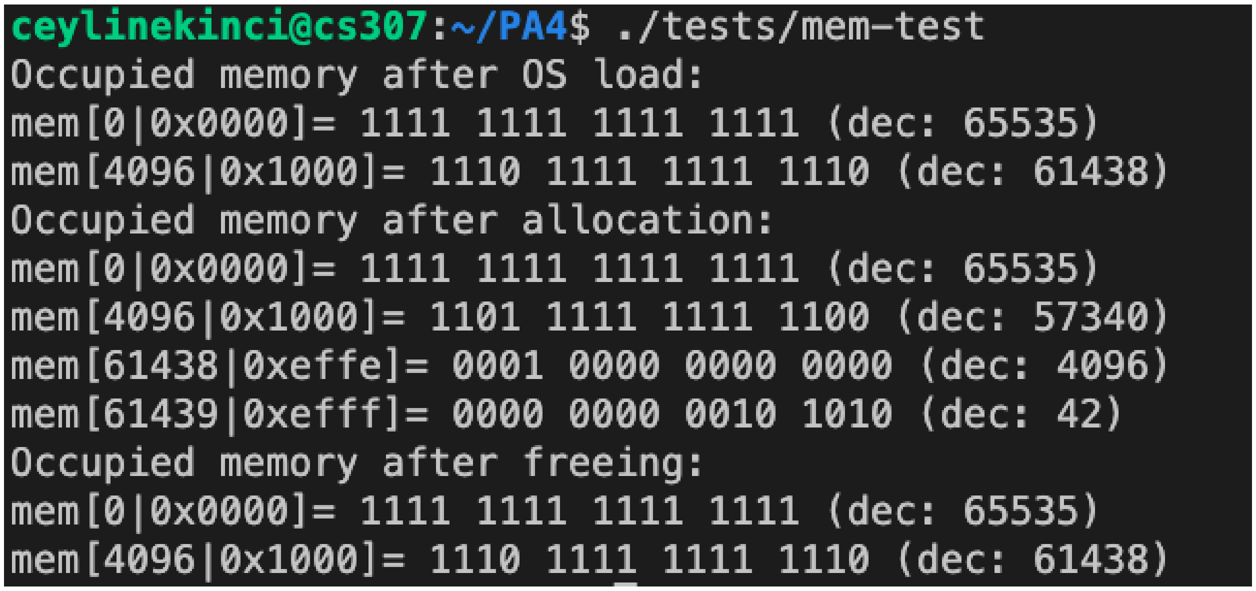
The createProc function creates a new process by allocating memory for its code and heap segments, initializing its PCB, and loading the code and heap segments from object files.The loadProc function loads a process by setting its PC, base, and bound registers from its PCB. Proc-test works correctly.



```
ceylinekinci@cs307:~/PA4$ ./tests/proc-test
Occupied memory after OS load:
mem[0|0x0000]= 1111 1111 1111 1111 (dec: 65535)
mem[4096|0x1000]= 1110 1111 1111 1110 (dec: 61438)
Occupied memory after program load:
mem[0|0x0000]= 1111 1111 1111 1111 (dec: 65535)
mem[1|0x0001]= 0000 0000 0000 0001 (dec: 1)
mem[13|0x000d]= 0011 0000 0000 0000 (dec: 12288)
mem[14|0x000e]= 1111 0000 0000 0000 (dec: 61440)
mem[15|0x000f]= 0001 0000 0000 0000 (dec: 4096)
mem[16|0x0010]= 1101 1111 1111 1110 (dec: 57342)
mem[17|0x0011]= 0001 0000 0000 0000 (dec: 4096)
mem[4096|0x1000]= 1100 1111 1111 1010 (dec: 53242)
mem[57340|0xdffc]= 0001 0000 0000 0000 (dec: 4096)
mem[57341|0xdffd]= 0000 0000 0010 1010 (dec: 42)
mem[57342|0xdffe]= 0000 0000 0000 0101 (dec: 5)
mem[57343|0xdfff]= 0000 0000 0000 0010 (dec: 2)
mem[57344|0xe000]= 0000 0000 0000 0001 (dec: 1)
mem[57345|0xe001]= 0000 0000 0000 0010 (dec: 2)
mem[57346|0xe002]= 0000 0000 0000 0011 (dec: 3)
mem[57347|0xe003]= 0000 0000 0000 0001 (dec: 1)
mem[57348|0xe004]= 0000 0000 0000 0010 (dec: 2)
mem[57349|0xe005]= 0000 0000 0000 0001 (dec: 1)
mem[57350|0xe006]= 0000 0000 0000 0010 (dec: 2)
mem[57351|0xe007]= 0000 0000 0000 0001 (dec: 1)
mem[61438|0xeffe]= 0001 0000 0000 0000 (dec: 4096)
mem[61439|0xefff]= 0000 0000 0010 1010 (dec: 42)
mem[61440|0xf000]= 0101 0010 0110 0000 (dec: 21088)
mem[61441|0xf001]= 0101 1001 0010 0000 (dec: 22816)
mem[61442|0xf002]= 0001 1001 0010 1010 (dec: 6442)
mem[61443|0xf003]= 1110 0100 0000 1000 (dec: 58376)
mem[61444|0xf004]= 0110 0100 1000 0000 (dec: 25728)
mem[61445|0xf005]= 0110 0110 1000 0000 (dec: 26240)
mem[61446|0xf006]= 0001 0100 1010 0001 (dec: 5281)
mem[61447|0xf007]= 0001 0010 0100 0011 (dec: 4675)
mem[61448|0xf008]= 0001 1001 0011 1111 (dec: 6463)
mem[61449|0xf009]= 0000 0011 1111 1011 (dec: 1019)
mem[61450|0xf00a]= 1111 0000 0010 1000 (dec: 61480)
mem[61451|0xf00b]= 1111 0000 0010 0101 (dec: 61477)
mem[61452|0xf00c]= 0100 0000 0000 0000 (dec: 16384)
```

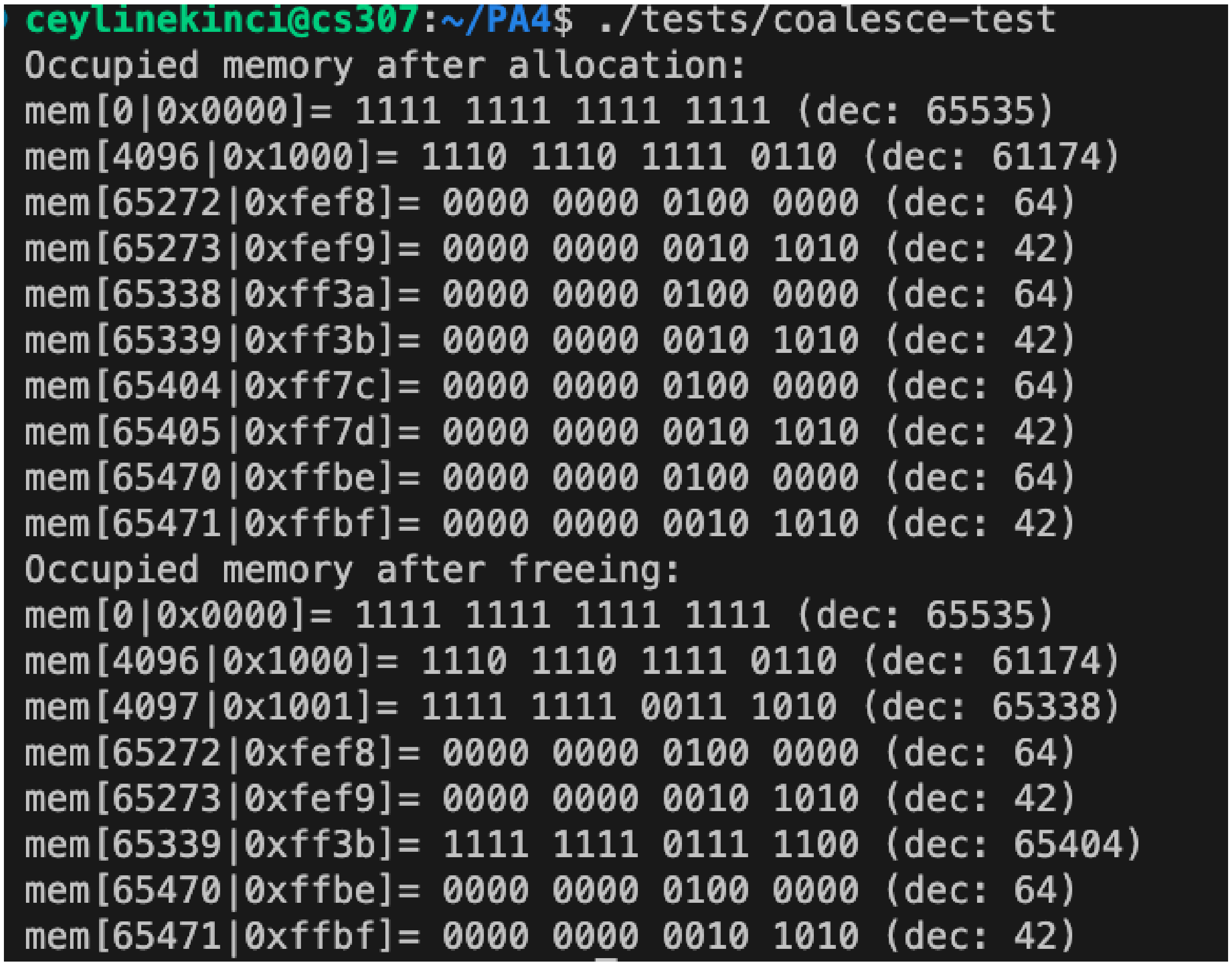
3. Memory Allocation and Deallocation (allocMem and freeMem)

The allocMem function allocates memory by iterating over free chunks and selecting the first chunk with enough space. The freeMem function frees memory and performs coalescing if adjacent chunks are also free. Mem-Test works correctly.



```
ceylinekinci@cs307:~/PA4$ ./tests/mem-test
Occupied memory after OS load:
mem[0|0x0000]= 1111 1111 1111 1111 (dec: 65535)
mem[4096|0x1000]= 1110 1111 1111 1110 (dec: 61438)
Occupied memory after allocation:
mem[0|0x0000]= 1111 1111 1111 1111 (dec: 65535)
mem[4096|0x1000]= 1101 1111 1111 1100 (dec: 57340)
mem[61438|0xeffe]= 0001 0000 0000 0000 (dec: 4096)
mem[61439|0xefff]= 0000 0000 0010 1010 (dec: 42)
Occupied memory after freeing:
mem[0|0x0000]= 1111 1111 1111 1111 (dec: 65535)
mem[4096|0x1000]= 1110 1111 1111 1110 (dec: 61438)
```

Coalesce-test also works correctly:

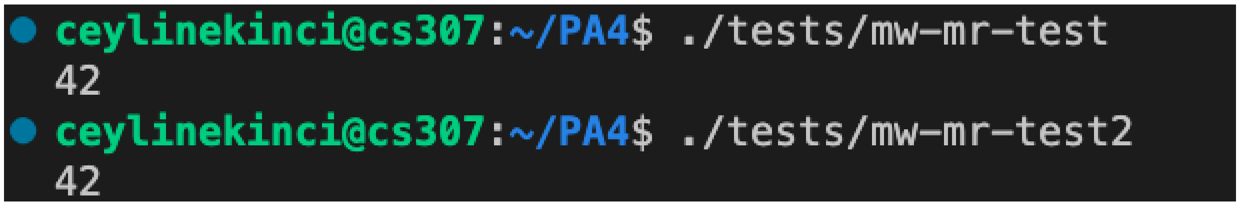


```
ceylinekinci@cs307:~/PA4$ ./tests/coalesce-test
Occupied memory after allocation:
mem[0|0x0000]= 1111 1111 1111 1111 (dec: 65535)
mem[4096|0x1000]= 1110 1110 1111 0110 (dec: 61174)
mem[65272|0xfef8]= 0000 0000 0100 0000 (dec: 64)
mem[65273|0xfef9]= 0000 0000 0010 1010 (dec: 42)
mem[65338|0xff3a]= 0000 0000 0100 0000 (dec: 64)
mem[65339|0xff3b]= 0000 0000 0010 1010 (dec: 42)
mem[65404|0xff7c]= 0000 0000 0100 0000 (dec: 64)
mem[65405|0xff7d]= 0000 0000 0010 1010 (dec: 42)
mem[65470|0xffbe]= 0000 0000 0100 0000 (dec: 64)
mem[65471|0xffbf]= 0000 0000 0010 1010 (dec: 42)
Occupied memory after freeing:
mem[0|0x0000]= 1111 1111 1111 1111 (dec: 65535)
mem[4096|0x1000]= 1110 1110 1111 0110 (dec: 61174)
mem[4097|0x1001]= 1111 1111 0011 1010 (dec: 65338)
mem[65272|0xfef8]= 0000 0000 0100 0000 (dec: 64)
mem[65273|0xfef9]= 0000 0000 0010 1010 (dec: 42)
mem[65339|0xff3b]= 1111 1111 0111 1100 (dec: 65404)
mem[65470|0xffbe]= 0000 0000 0100 0000 (dec: 64)
mem[65471|0xffbf]= 0000 0000 0010 1010 (dec: 42)
```

4. Modifying mr and mw Functions

The mr and mw functions now perform address translation and segmentation fault checks.

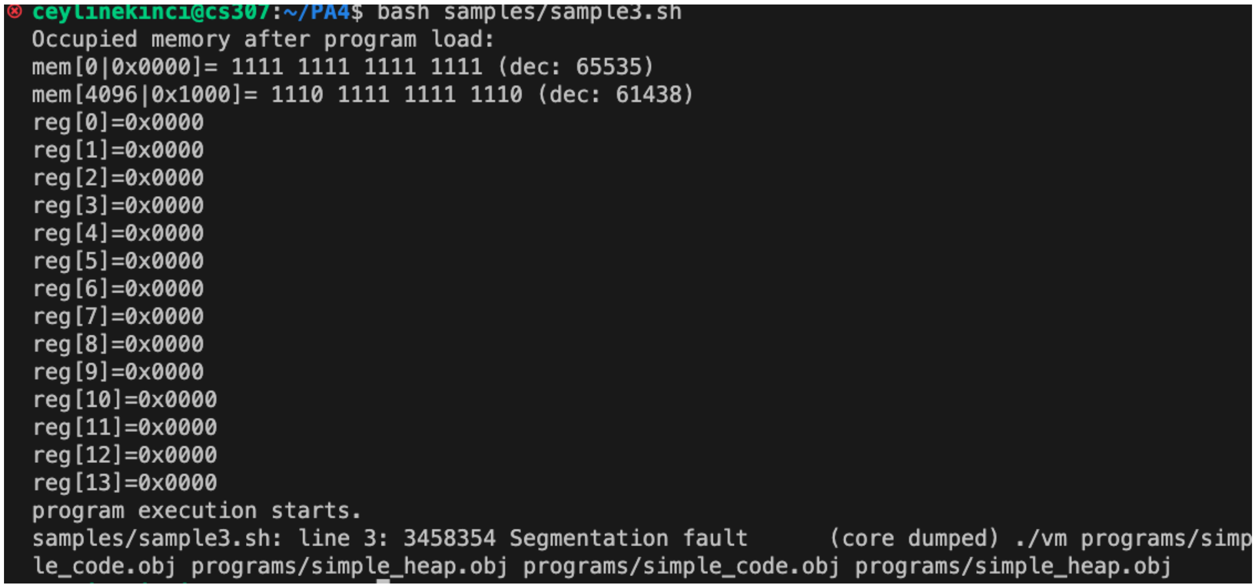
Test1 works correctly but second is not right.



```
● ceylinekinci@cs307:~/PA4$ ./tests/mw-mr-test
  42
● ceylinekinci@cs307:~/PA4$ ./tests/mw-mr-test2
  42
```

5. Implementing tyld and thalt Trap Instructions

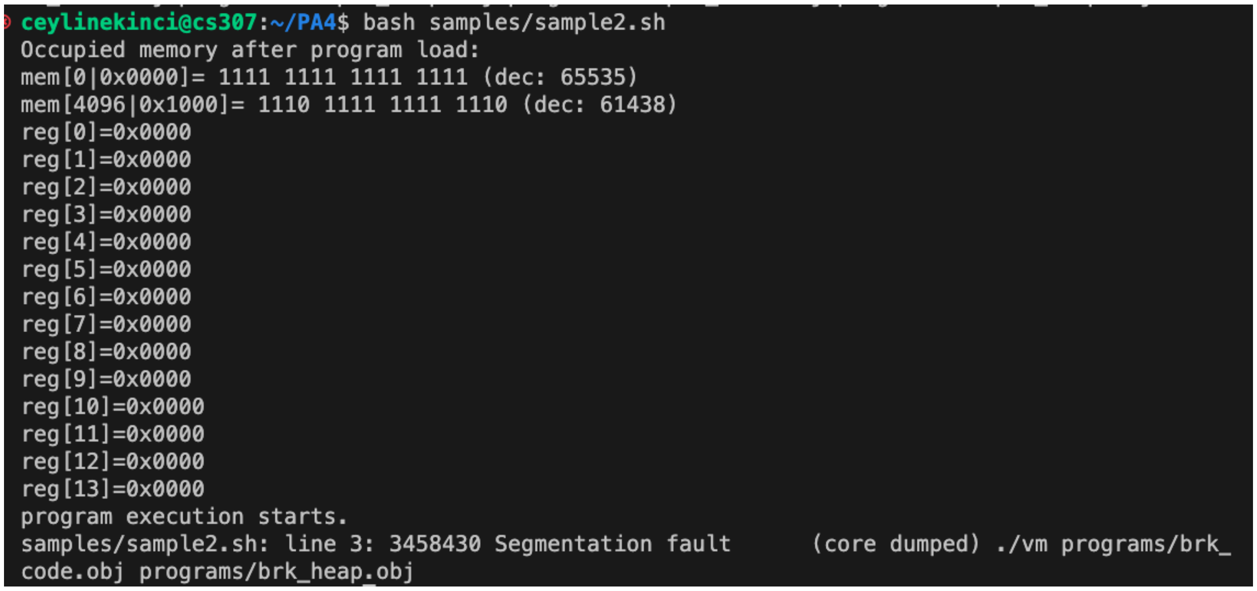
The tyld instruction allows voluntary context switching, while the thalt instruction handles process termination. Not working correctly.



```
⊗ ceylinekinci@cs307:~/PA4$ bash samples/sample3.sh
  Occupied memory after program load:
  mem[0|0x0000]= 1111 1111 1111 1111 (dec: 65535)
  mem[4096|0x1000]= 1110 1111 1111 1110 (dec: 61438)
  reg[0]=0x0000
  reg[1]=0x0000
  reg[2]=0x0000
  reg[3]=0x0000
  reg[4]=0x0000
  reg[5]=0x0000
  reg[6]=0x0000
  reg[7]=0x0000
  reg[8]=0x0000
  reg[9]=0x0000
  reg[10]=0x0000
  reg[11]=0x0000
  reg[12]=0x0000
  reg[13]=0x0000
  program execution starts.
  samples/sample3.sh: line 3: 3458354 Segmentation fault      (core dumped) ./vm programs/simp
  le_code.obj programs/simple_heap.obj programs/simple_code.obj programs/simple_heap.obj
```

6. Implementing the tbrk Trap Instruction

The tbrk instruction handles resizing the heap segment of the currently running process. Not working correctly.



```
ceylinekinci@cs307:~/PA4$ bash samples/sample2.sh
Occupied memory after program load:
mem[0|0x0000]= 1111 1111 1111 1111 (dec: 65535)
mem[4096|0x1000]= 1110 1111 1111 1110 (dec: 61438)
reg[0]=0x0000
reg[1]=0x0000
reg[2]=0x0000
reg[3]=0x0000
reg[4]=0x0000
reg[5]=0x0000
reg[6]=0x0000
reg[7]=0x0000
reg[8]=0x0000
reg[9]=0x0000
reg[10]=0x0000
reg[11]=0x0000
reg[12]=0x0000
reg[13]=0x0000
program execution starts.
samples/sample2.sh: line 3: 3458430 Segmentation fault      (core dumped) ./vm programs/brk_
code.obj programs/brk_heap.obj
```