COMP1521 Tutorial 07

File Seeking

- Lseek(fd, offset, whence) is a system call that allows us to change the position of the file position (invisible cursor) in an open file
- It returns -1 if it fails, or the position in the file (bytes) if successful
- When would it be useful?

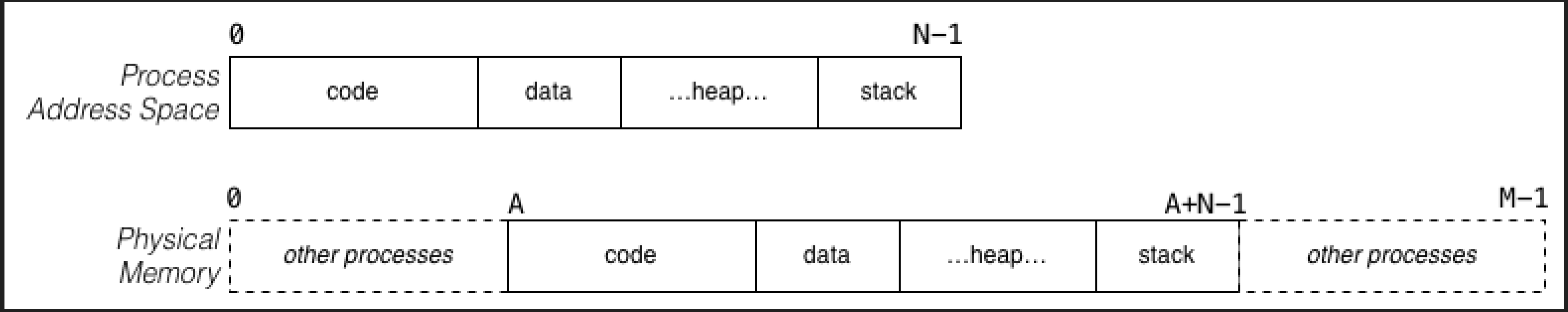
File Seeking

Consider a file of size 10000 bytes, open for reading on file descriptor fd, initially positioned at the start of the file (offset 0). What will be the file position after each of these calls to lseek()? Assume that they are executed in sequence and one will change the file state that the next one deals with

1. lseek(fd, 0, SEEK_END);
2. lseek(fd, -1000, SEEK_CUR);
3. lseek(fd, 0, SEEK_SET);
4. lseek(fd, -100, SEEK_SET);
5. lseek(fd, 1000, SEEK_SET);
6. lseek(fd, 1000, SEEK_CUR);

Memory Management

- Processes used to be stored in memory as one contiguous (one after another) chunk but doesn't always start from 0



- This will require the program to be rewritten but with new addresses relative to the base address

Memory Management

Consider the following piece of MIPS code,

- loop1 is located at 0x1000
- end_loop1 is located at 0x1028
- array is located at 0x2000.

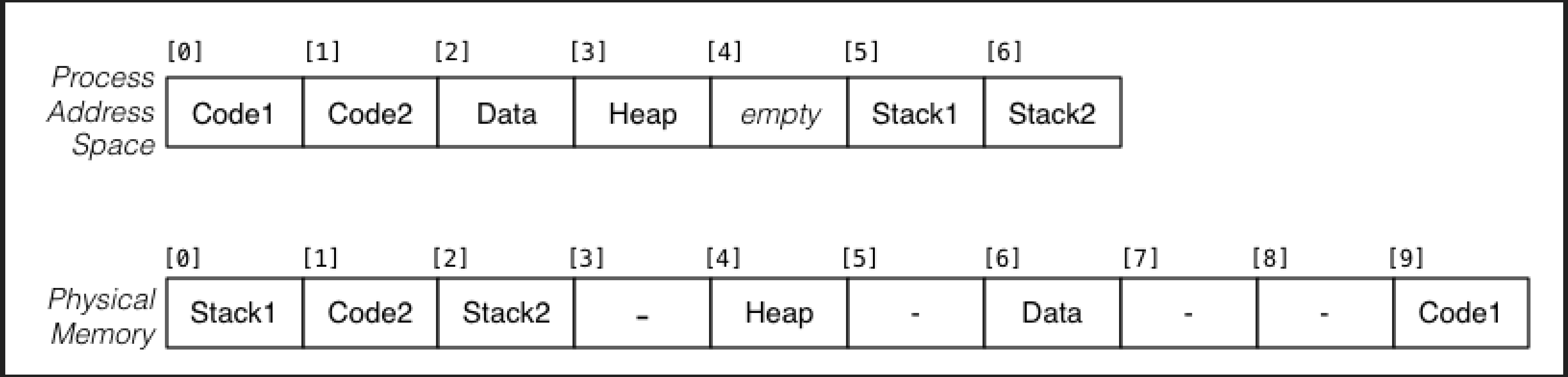
If the program containing this code is loaded starting at address A = 0x8000, which instructions need to be rewritten and what addresses are in the relocated code?

```
   li  $t0, 0
   li  $t1, 0
   li  $t2, 20  # elements in array
loop1:
   bge $t1, $t2, end_loop1
   mul $t3, $t1, 4
   la  $t4, array
   add $t3, $t3, $t4
   lw  $t3, ($t3)
   add $t1, $t1, $t3
   add $t1, $t1, 1
   j   loop1
end_loop1:
```

Virtual Memory System

- Efficient system primarily used now in modern operating systems to manage memory
- Physical memory is split into equal chunks called frames
    - Memory addresses in physical memory are called **physical addresses**
- Frames are allocated to the page table
- Page table keeps track of which frames belong to which process creating an *Address Space* for each
    - Addresses within the *Address Space* are called **virtual addresses**
- Allows programs to only be partially loaded into memory

Virtual Memory



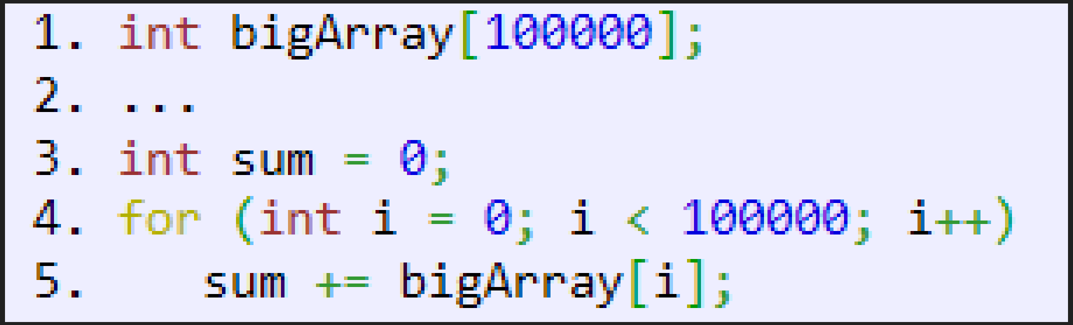
For each of the following process addresses (in decimal notation), determine what physical address it maps to.

- jal func, where the label func is at 5096
- lw $s0,($sp), where $sp contains 28668
- la $t0, msg where the label msg is at 10192

Working Set

- The *working set* are a set of pages currently being used by processes
  - Code being executed
  - Data associated with the code being executed at that time

Working Set



```
int bigArray[100000];
...
int sum = 0;
for (int i = 0; i < 100000; i++)
   sum += bigArray[i];
```

Assuming pages are 4KB, all code here fits in one page and there is one process:

- How large is the working set of this piece of code?
- Assuming that the code is already loaded in memory, but that none of bigArray is loaded, and that only the working set is held in memory, how many page faults are likely to be generated during the execution of this code?

Working Set

- https://cgi.cse.unsw.edu.au/~cs1521/18s2/tutors/tutes/week07/index.php##