CSE 451: Section 4 Handout
10/17/2019

Pipes and Exec

Pipes:

If a pipe no longer has a reader, a `write` call should return an error.
If a pipe no longer has a writer and the buffer is empty, a `read` should return a 0.

What can `read` return if the pipe no longer has a writer, but the buffer has data?

Programs do **not** normally call close on a pipe when they are done.
It can be the case that a parent and child are communicating through a pipe.
The parent is the reader, and the child is the writer.

If the parent is sitting in a loop reading information from the pipe, how can you ensure that the parent will get a 0 return from a `read` call when the child has finished executing?

Imagine a reader of a pipe is sleeping waiting for the writer to write some data and wake them up.
If the writer process is killed before it actually gets a chance to writer data, how does the reader get woken up?
What should the read call from the reader return once it's woken up?

CSE 451: Section 4 Handout
10/17/2019

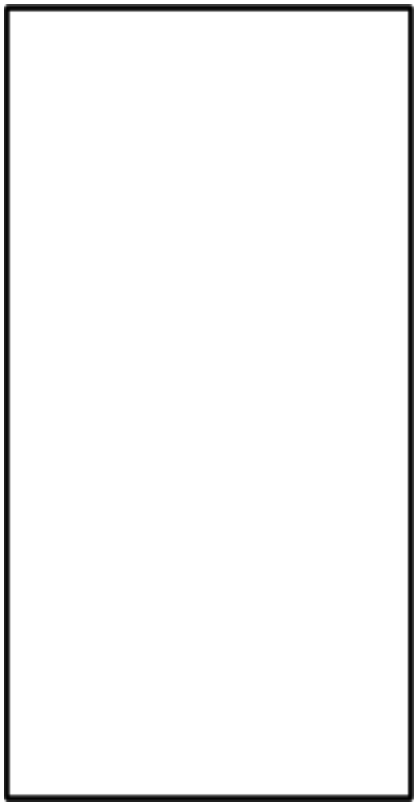
Pipes and Exec

Exec:

When creating the user stack in xk, what should the stack pointer (rsp) start at?
Note: This would be an argument to pass to `vspaceinitstack`

What does the virtual address space stack look like after the following exec call?**:**

```
char *echo_args[] = [“echo”, “hello”, “world”, “\0”];
exec(“echo”, echo_args);
```



What are the register values (rdi, rsi, rsp)?

CSE 451: Section 4 Handout
10/17/2019

Pipes and Exec

`vspaceinstall` takes a pointer to a process and loads the page table register (cr3) with the address of the process's page directory.
This is required for the cpu to allow the process to execute using virtual memory.

exec should change the current running process's virtual address space.
When should `vspaceinstall` be called in exec?

Consider two binaries `ayy` and `lmao` that were generated from the following source programs

ayy.c:
```
int main(int argc, char** argv) {
  // Print space-separated arguments.
  for (int i = 0; i < argc; i++) {
    printf("%s ", argv[i]);
  }
  printf("\n");
  if (argc == 1) {
    exec("ayy", ["ayy", "useful", "this", "time", 0]);
  } else if (argc == 2) {
    exec("lmao", ["lmao", "done", 0]);
  } else {
    exec("lmao", ["lmao", "just", "sayin", "hi", 0]);
  }

  return 0;
}
```

lmao.c:
```
int main(int argc, char** argv) {
  // Print space-separated arguments.
  for (int i = 0; i < argc; i++) {
    printf("%s ", argv[i]);
  }
  printf("\n");
  if (argc != 2) {
    exec("ayy", ["ayy", "return", 0]);
  }
  return 0;
}
```

What gets printed if we run `exec("ayy", ["ayy", 0])`?