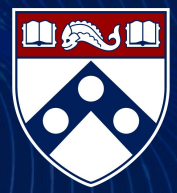


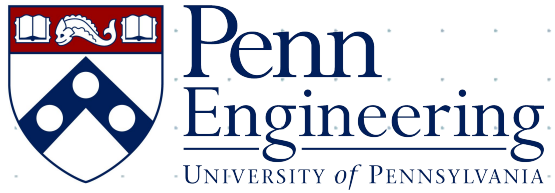
Penn Engineering
University of Pennsylvania

CIS 3990 Recitation 05

Git and Processes

2025-10-02

Agenda

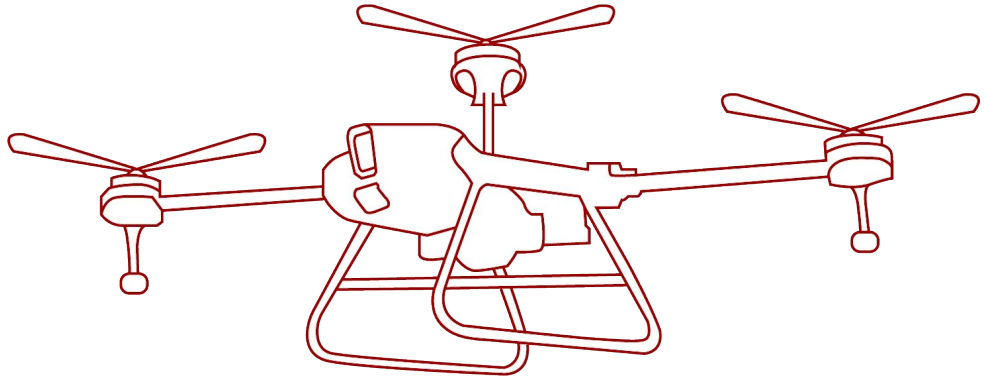


01 Logistics

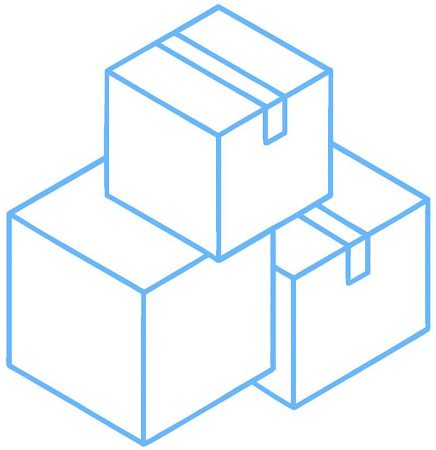
02 Processes

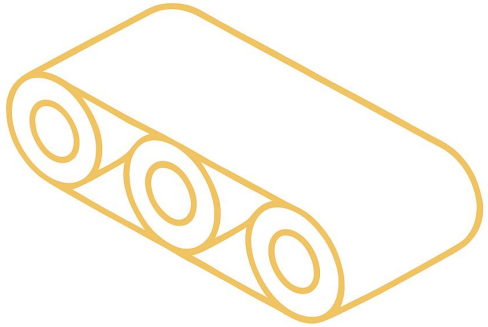
03 Locality

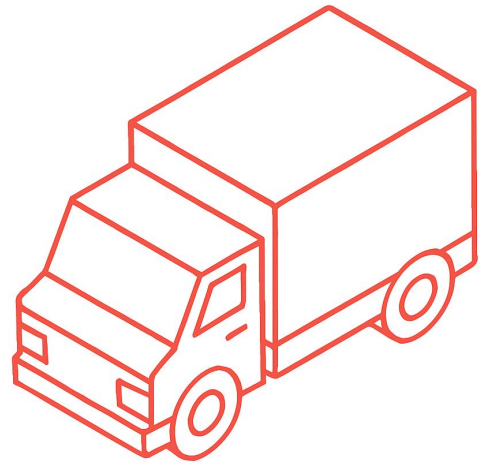
Penn Engineering
University of Pennsylvania



Logistics



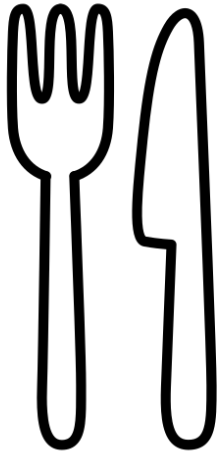




Logistics

- HW05 out - due 11:59pm on Tues, Oct 7 - via Gradescope
- Check-In 04 is posted
- Don’t forget to use check-ins to reopen past homeworks
  - Finish the unfinished
  - Fix style (HW1 and HW2)

Penn Engineering
University of Pennsylvania



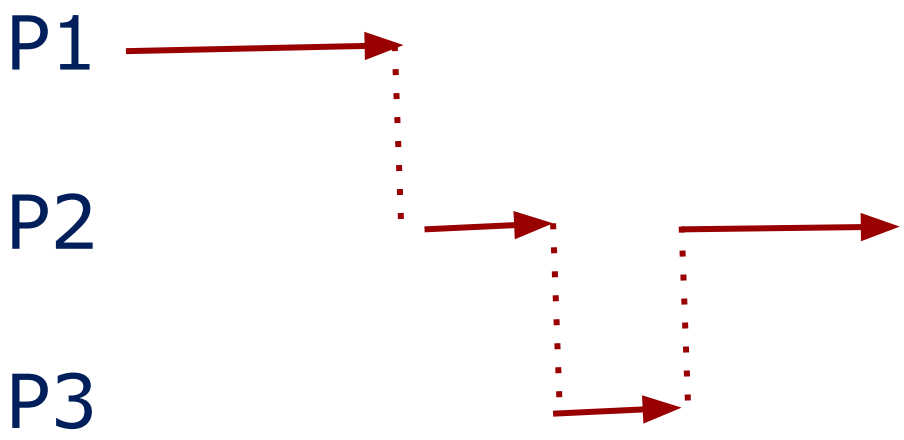
Processes

fork, exec

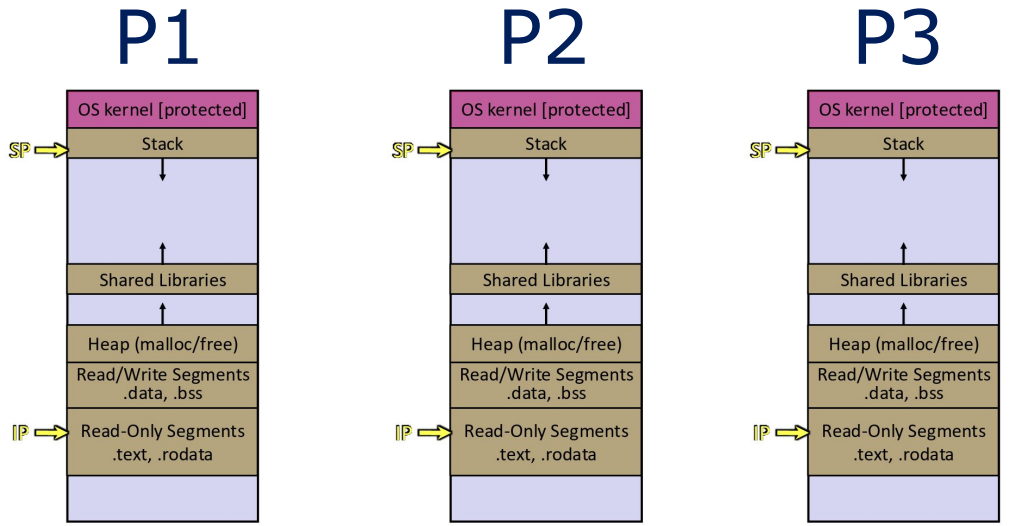
Processes - the mental model

- Two ways of viewing a process (or multiple processes in interleaved execution)

1. line(s) of execution

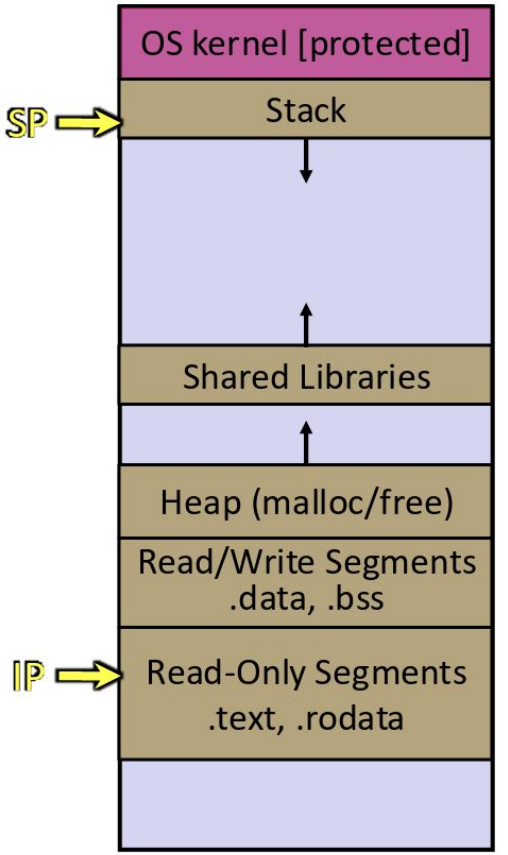


2. separate memory environments



Fork

- Process that calls fork() has its entire memory copied by OS
  - "main memory" aka stack, heap
  - read-only and read-write memory
    - readonly contains our code, which our IP points to
- not seen:
  - registers (including instruction pointer)
  - file descriptors
  - and more!
- returns child's pid to the parent, and 0 to the child



Debate

- Would you say that fork is the only function that "returns twice?"

- Why or why not?

Exercise 1

Penn Engineering
University of Pennsylvania

Which of the following can the child modify, such that the can parent observe these changes? Assume the parent had access to these before calling fork().

a. local variables
b. static variables
c. global variables
d. read-write memory
e. file descriptors
f. files

Exercise 2

How many different combinations are possible?

```
int main(void){
  int level_1 = fork();
  if (level_1 == 0) {
    int level_2a = fork();
    if (level_2a == 0) {
      cout << "A" << endl;
    } else {
      cout << "B" << endl;
    }
  } else {
    int level_2b = fork();
    if (level_2b != 0) {
      cout << "C" << endl;
      exit(0);
    }
    cout << "D" << endl;
  }
  cout << "0" << endl;
  return (0);
}
```

Exercise 3

Penn Engineering
University of Pennsylvania

What does this print?

```
#include <iostream>
#include <unistd.h>
#include <cstdlib>

int main() {
 int res = fork();
   if (res == 0) {
       write(1, "hello", 5);
   }
   res = fork();
   if (res == 0) {
       write(1, "hello", 5);
   }
   return EXIT_SUCCESS;
}
```

Exercise 4

Penn Engineering
University of Pennsylvania

What does this print?

```
#include <iostream>
#include <unistd.h>
#include <cstdlib>
using namespace std;

int main() {
   int res = fork();
   if (res == 0) {
       cout <<  "hello";
   }
   res = fork();
   if (res == 0) {
       cout << "hello";
   }
   return EXIT_SUCCESS;
}
```

What's the difference?

Penn Engineering
University of Pennsylvania

```
#include <iostream>
#include <unistd.h>
#include <cstdlib>


int main() {
 int res = fork();
   if (res == 0) {
       write(1, "hello", 5);
   }
   res = fork();
   if (res == 0) {
       write(1, "hello", 5);
   }
   return EXIT_SUCCESS;
}
```

```
#include <iostream>
#include <unistd.h>
#include <cstdlib>
using namespace std;

int main() {
   int res = fork();
   if (res == 0) {
       cout << "hello";
   }
   res = fork();
   if (res == 0) {
       cout << "hello";
   }
   return EXIT_SUCCESS;
}
```

write() versus cout

- write() and cout are both writing to the same file descriptor
- cout is line buffered - flushes when it encounters a newline or endl
  - exiting a program will flush all buffers too
- write is unbuffered - even if we are writing to stdout!
  - It sees stdout as just another file, doesn't know to treat it differently

How does this relate to fork()?

Penn Engineering
University of Pennsylvania

- fork() will copy all memory of the parent when it is creating the child

- buffers exist in memory, and any contents of the buffer will be copied on fork()

Tip: ensure all buffers are flushed before calling fork()

exec

- replace current process/program with another process/program
- resets all memory
- maintains open file descriptors*

- fork/exec pair is useful when you want to start a new program, but don’t want to kill the currently running one

Exercise 5

Penn Engineering
University of Pennsylvania

What does this print?

```
#include <iostream>
#include <unistd.h>

int main() {
    char *args[] = {(char *)"ls", (char *)"-l", (char
*)NULL};

    std::cout << "Before execvp()" << std::endl;

    // replaces process w/ ls -l
    execvp("ls", args);

    // executes only if execvp() fails
    std::cerr << "Exec failed!" << std::endl;
    return 1;
}
```


That's all Folks!