SER 334 A Session

SI Session

Monday, January 29th 2024

7:00 pm - 8:00 pm MST

Agenda

- Memory
  - Structure
  - Interprocess Communication
- Process Tracing
- Module 5 Samples

SI Session Expectations

Thanks for coming to the **SER 334** SI session. We have a packed agenda and we are going to try to get through as many of our planned example problems as possible. This session will be recorded and shared with others.

- If after this you want to see additional examples, please visit the drop-in tutoring center.
- We will post the link in the chat now and at the end of the session.
  - tutoring.asu.edu
- Please keep in mind we are recording this session and it will be made available for you to review 24-48 hours after this session concludes.
- Finally, please be respectful to each other during the session.

Interact with us:

Zoom Features

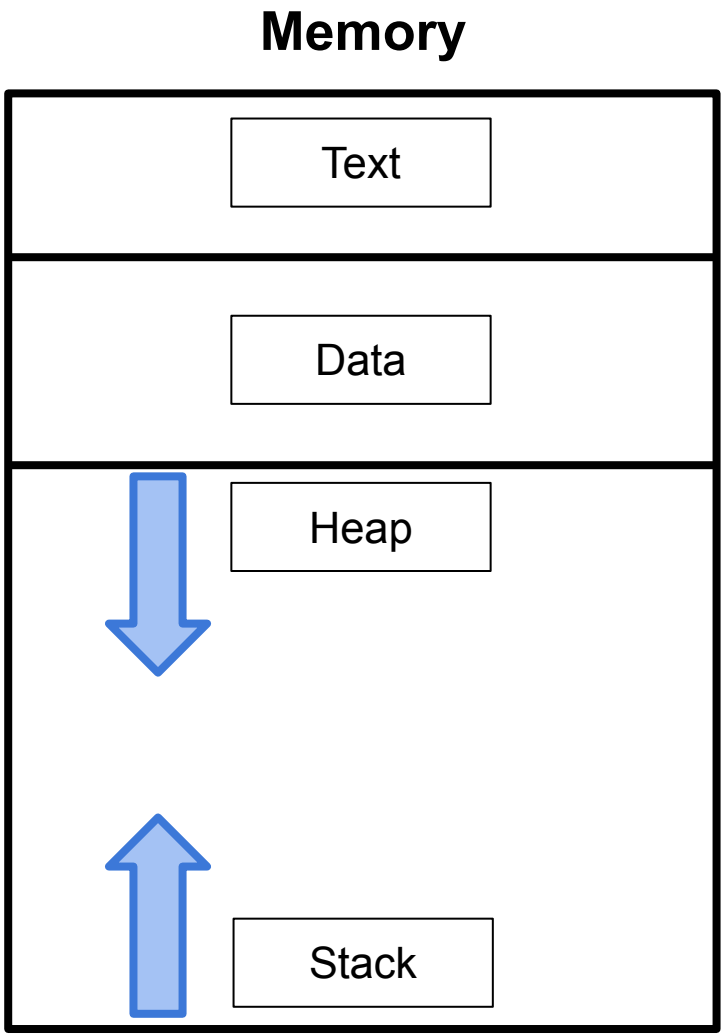
Zoom Chat

- Use the chat feature to interact with the presenter and respond to presenter's questions.
- Annotations are encouraged

SER 334

Memory

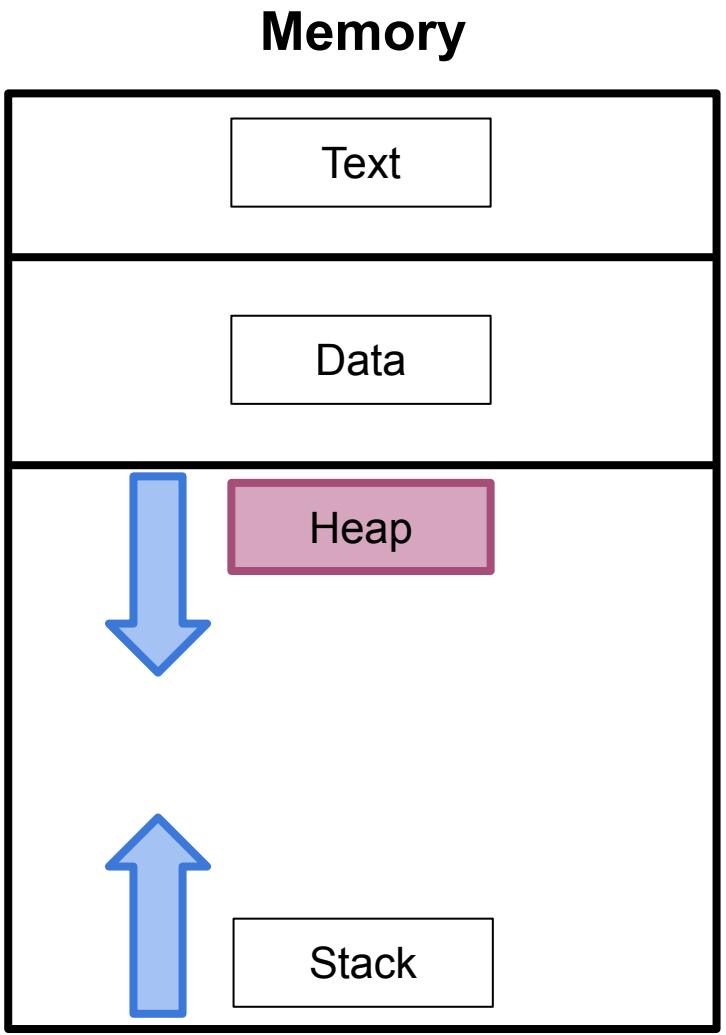
Which one holds dynamically allocated memory?



SER 334

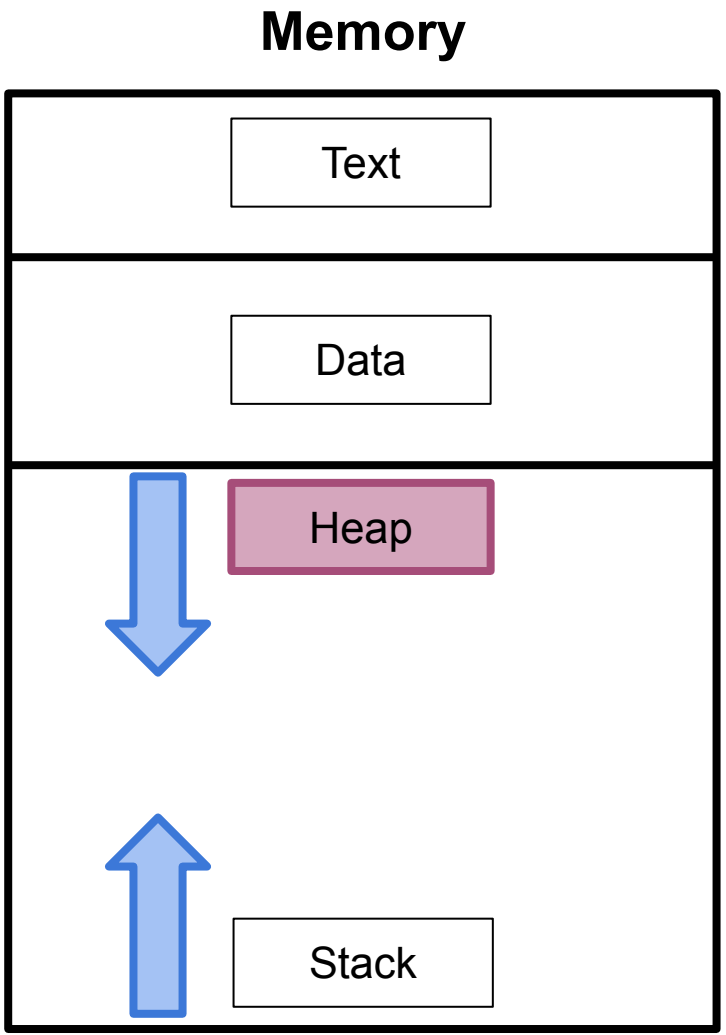
Memory

Which one holds dynamically allocated memory?



SER 334
Memory

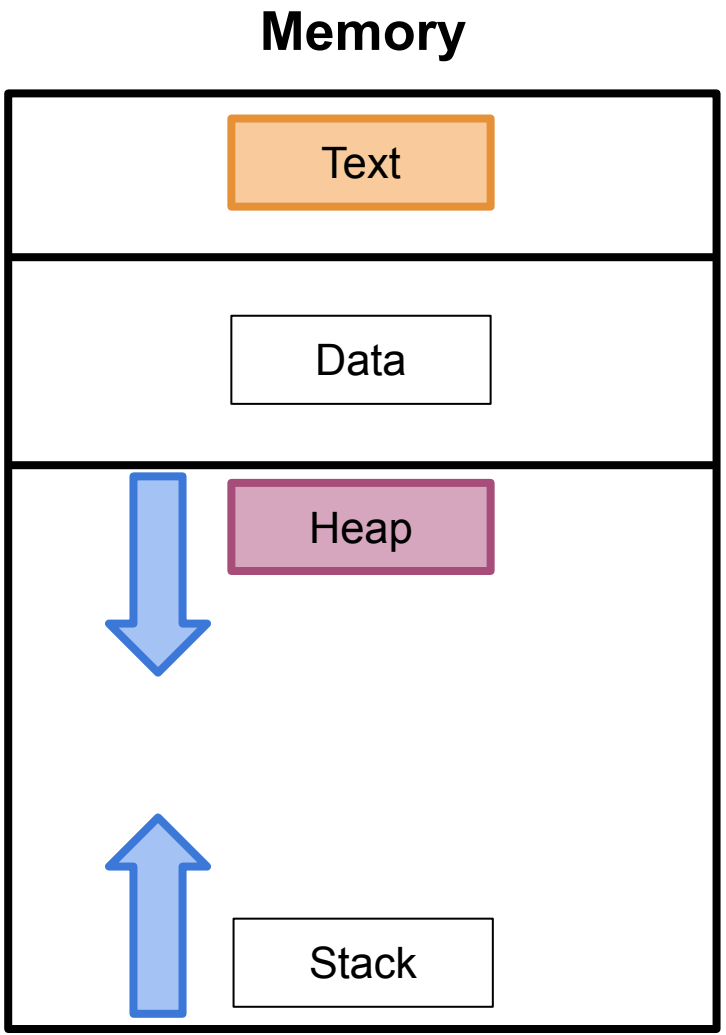
Which one holds our program code?



SER 334

Memory

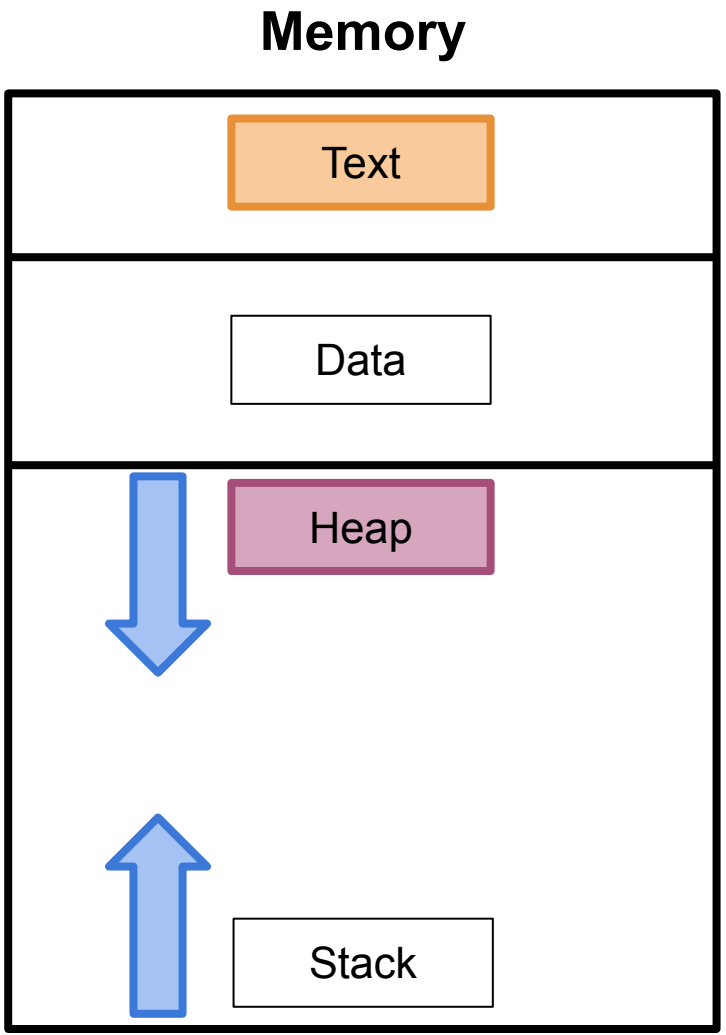
Which one holds our program code?



SER 334

Memory

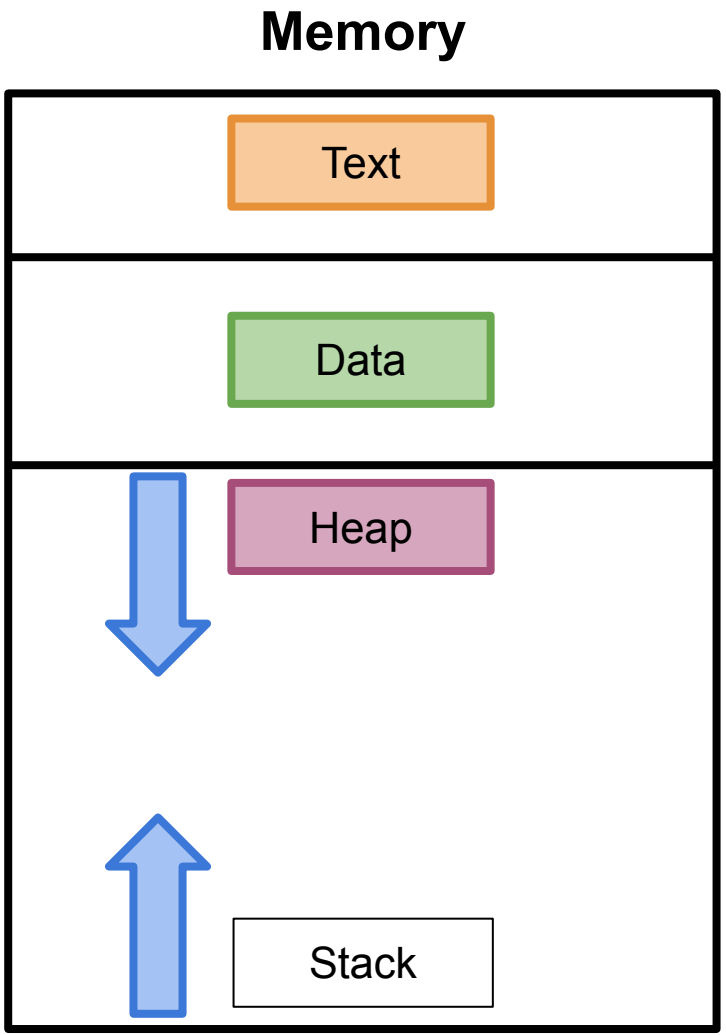
Which one holds global and static variables and objects?



SER 334

Memory

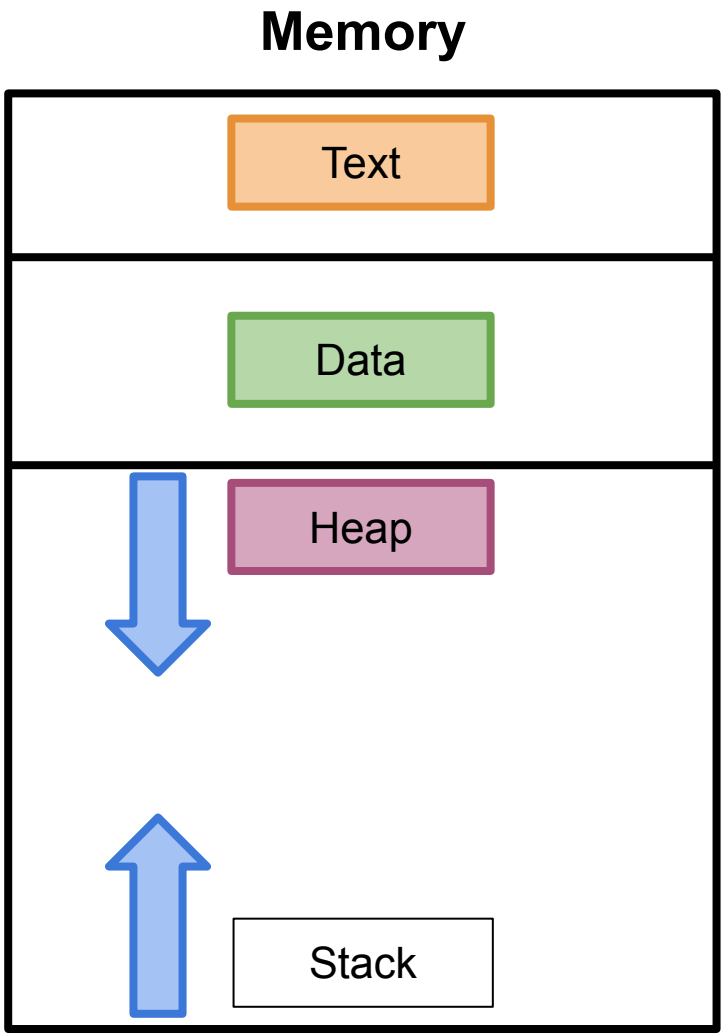
Which one holds global and static variables and objects?



SER 334

Memory

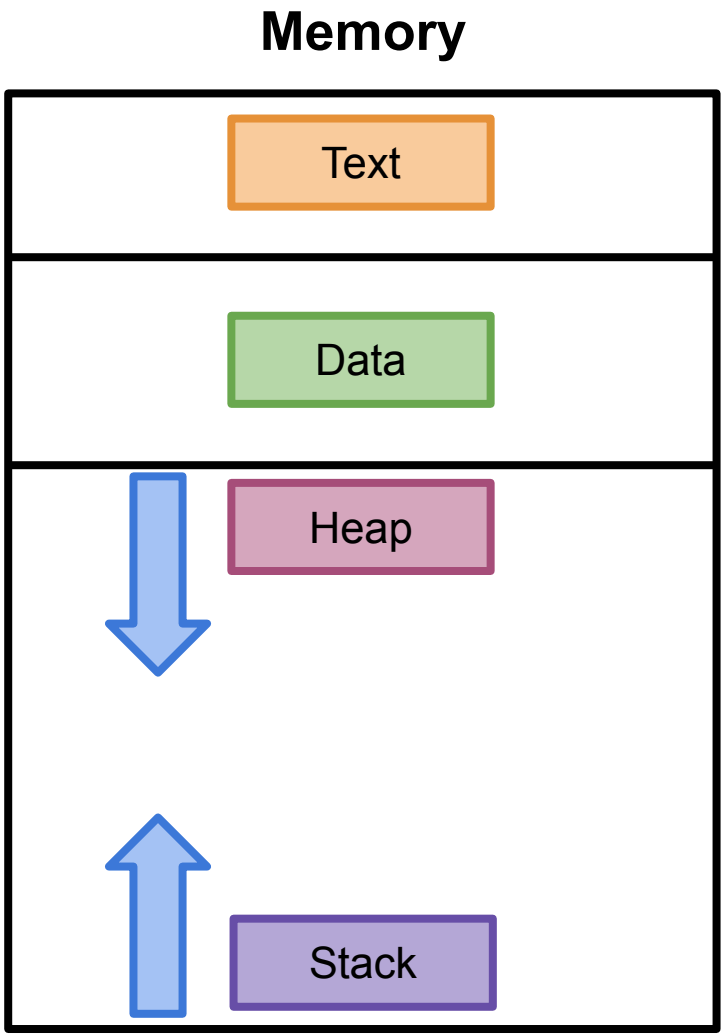
Which one holds temporary data?



SER 334

Memory

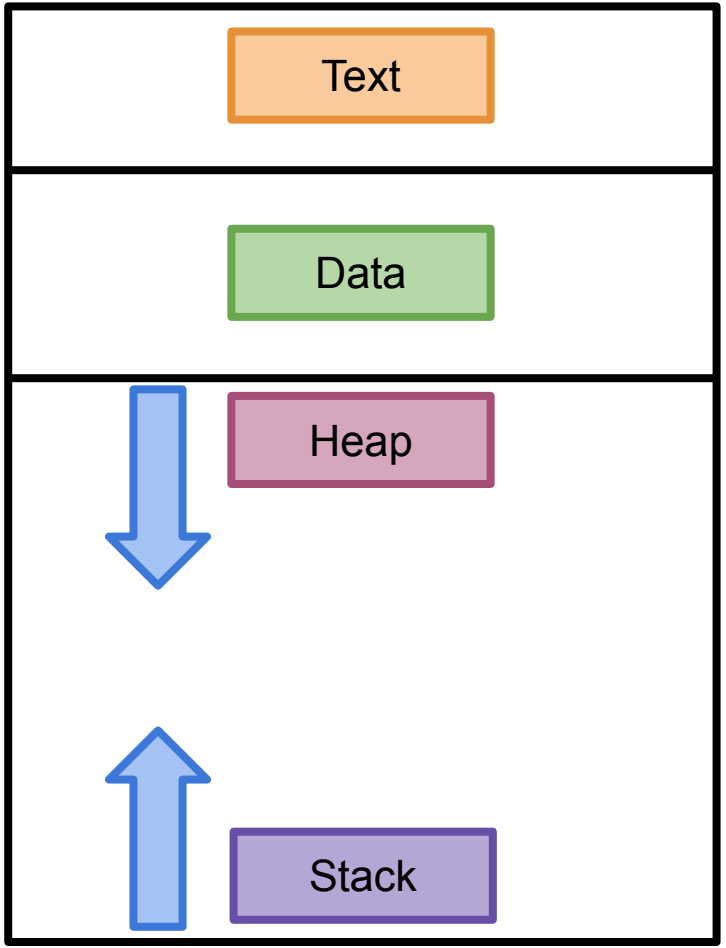
Which one holds temporary data?



SER 334
Memory

1. [Acuña] There are four general regions to a process in memory: stack, heap, data, and text. **List and explain which of these must be separated for each program running and which might be combined globally.** (Don't worry about security or programs misbehaving.) [2 points]

Memory



SER 334
Memory

Shared Memory

vs.

Message Passing

Module 5 Sample

SER 334
Processes

pid1

pid2

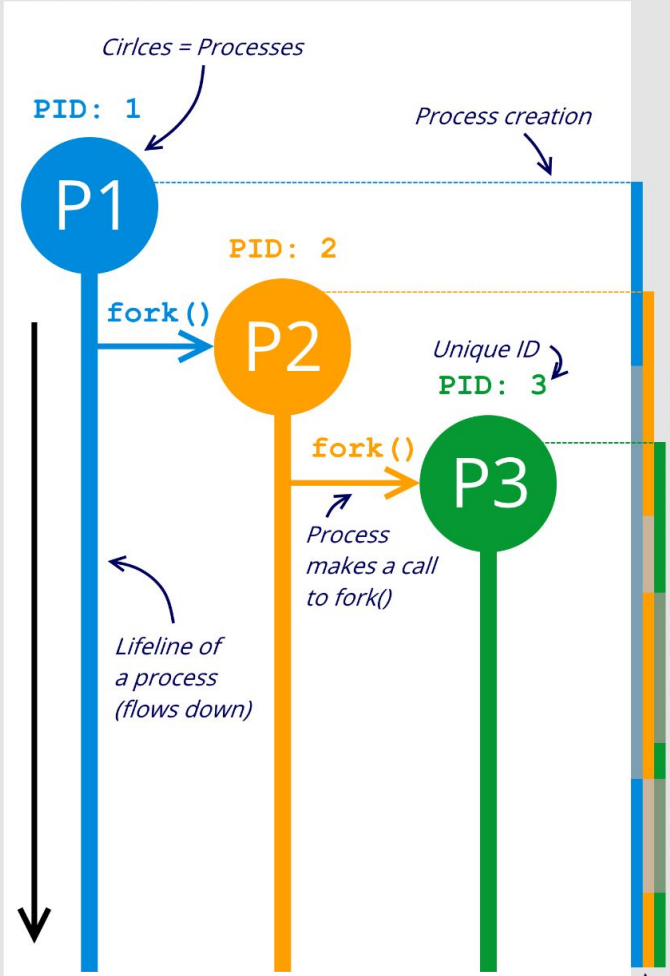
pid1

pid2

pid1

pid2

Output:




```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5  int main() {
6      pid_t pid1, pid2;
7      pid1 = fork();
8
9      if (pid1 == 0) {
10         printf("A");
11         pid2 = fork();
12
13         if (pid2 == 0) {
14             printf("B");
15         }
16         else {
17             wait(NULL);
18             printf("C");
19         }
20     }
21     else {
22         printf("D");
23     }
24     return 0;
25 }
```

Lines of code touched per each process

SER 334
Processes

pid1 2
pid2

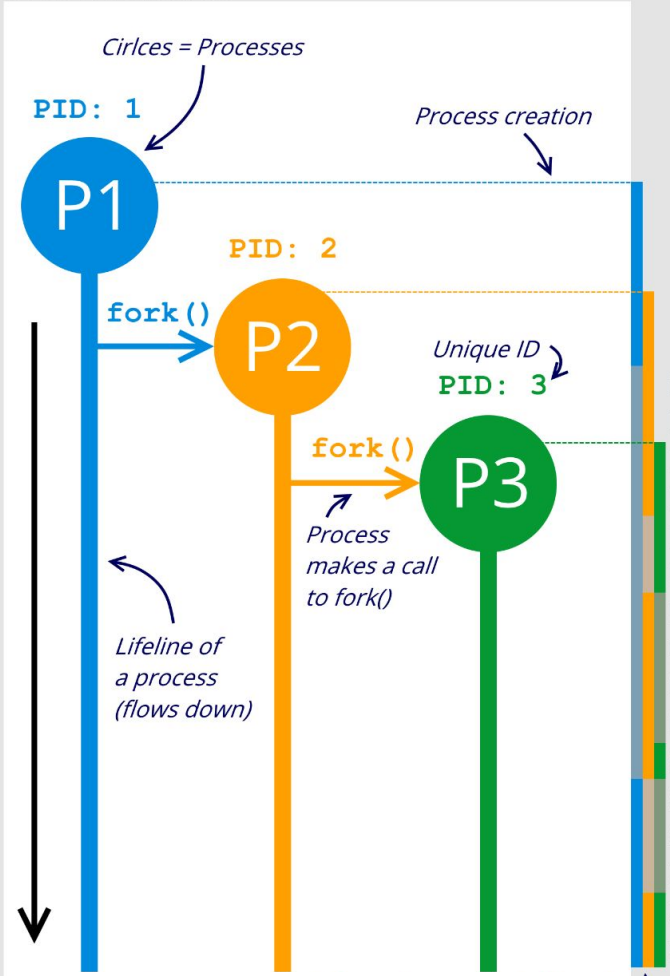
Why 2?

pid1
pid2

Why 0?

pid1
pid2

Output:




```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5  int main() {
6      pid_t pid1, pid2;
7      pid1 = fork();
8
9      if (pid1 == 0) {
10         printf("A");
11         pid2 = fork();
12
13         if (pid2 == 0) {
14             printf("B");
15         }
16         else {
17             wait(NULL);
18             printf("C");
19         }
20     }
21     else {
22         printf("D");
23     }
24     return 0;
25 }
```

Lines of code touched per each process

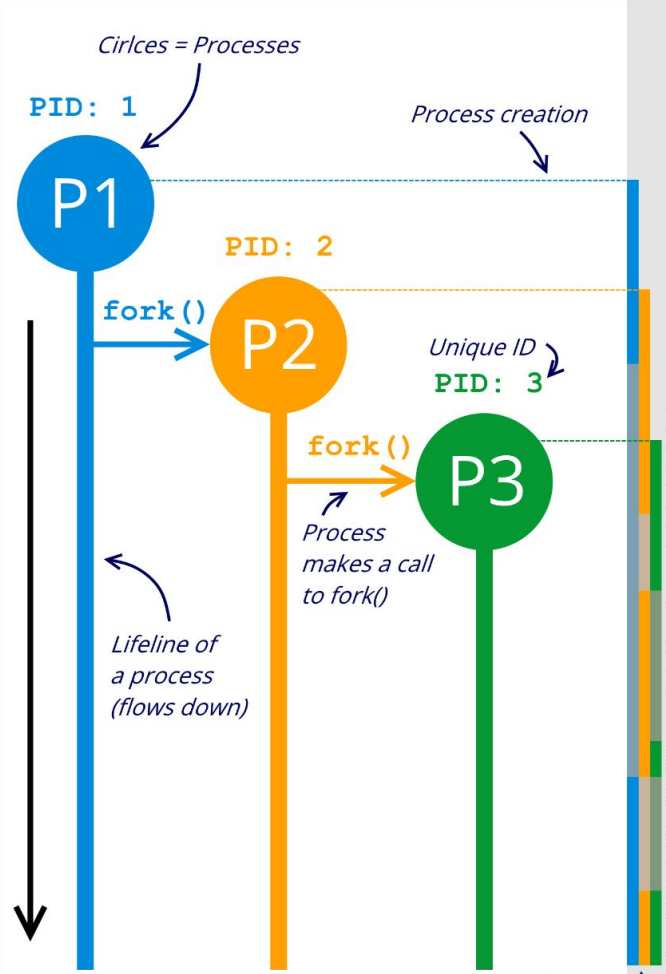
SER 334
Processes

pid1	2
pid2	

pid1	0
pid2	0

pid1	
pid2	

Output:




```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid1, pid2;
    pid1 = fork();

    if (pid1 == 0) {
        printf("A");
        pid2 = fork();

        if (pid2 == 0) {
            printf("B");
        }
        else {
            wait(NULL);
            printf("C");
        }
    }
    else {
        printf("D");
    }
    return 0;
}
```

Lines of code touched per each process

SER 334
Processes

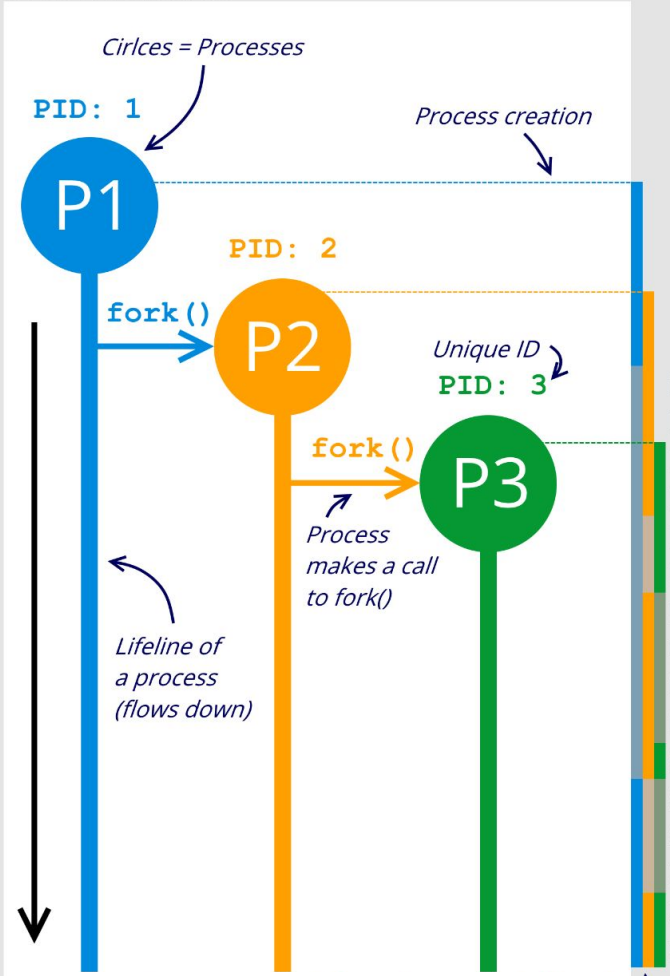
pid1	2
pid2	

pid1	0
pid2	0

pid1	
pid2	

Output:

A




```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5  int main() {
6      pid_t pid1, pid2;
7      pid1 = fork();
8
9      if (pid1 == 0) {
10         printf("A");
11         pid2 = fork();
12
13         if (pid2 == 0) {
14             printf("B");
15         }
16         else {
17             wait(NULL);
18             printf("C");
19         }
20     }
21     else {
22         printf("D");
23     }
24     return 0;
25 }
```

Lines of code touched per each process

SER 334
Processes

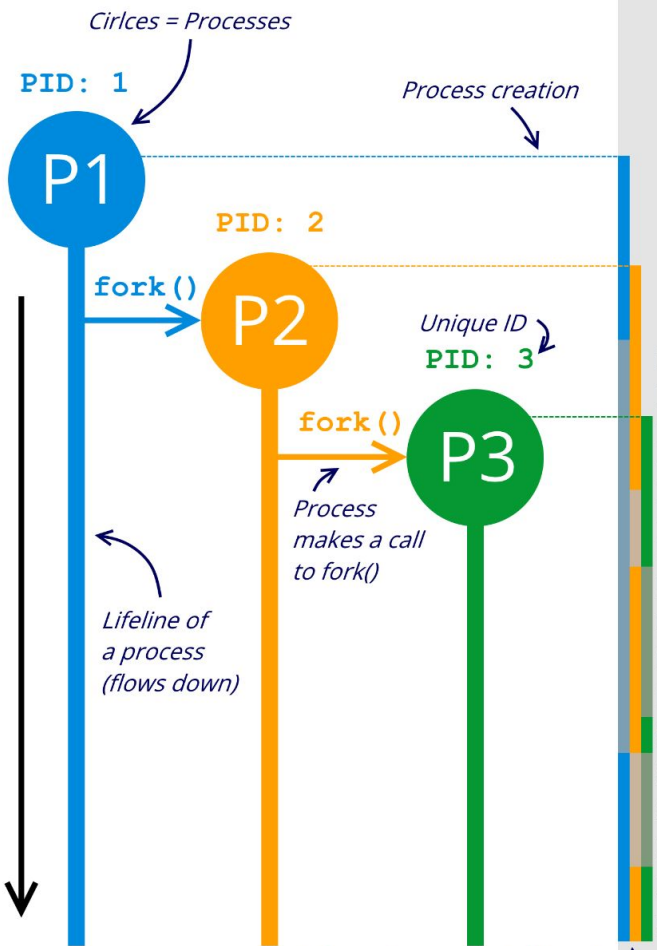
pid1	2
pid2	

pid1	0
pid2	3

pid1	0
pid2	0

Output:

A B




```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid1, pid2;
    pid1 = fork();

    if (pid1 == 0) {
        printf("A");
        pid2 = fork();

        if (pid2 == 0) {
            printf("B");
        }
        else {
            wait(NULL);
            printf("C");
        }
    }
    else {
        printf("D");
    }
    return 0;
}
```

Lines of code touched per each process

SER 334
Processes

pid1	2
pid2	

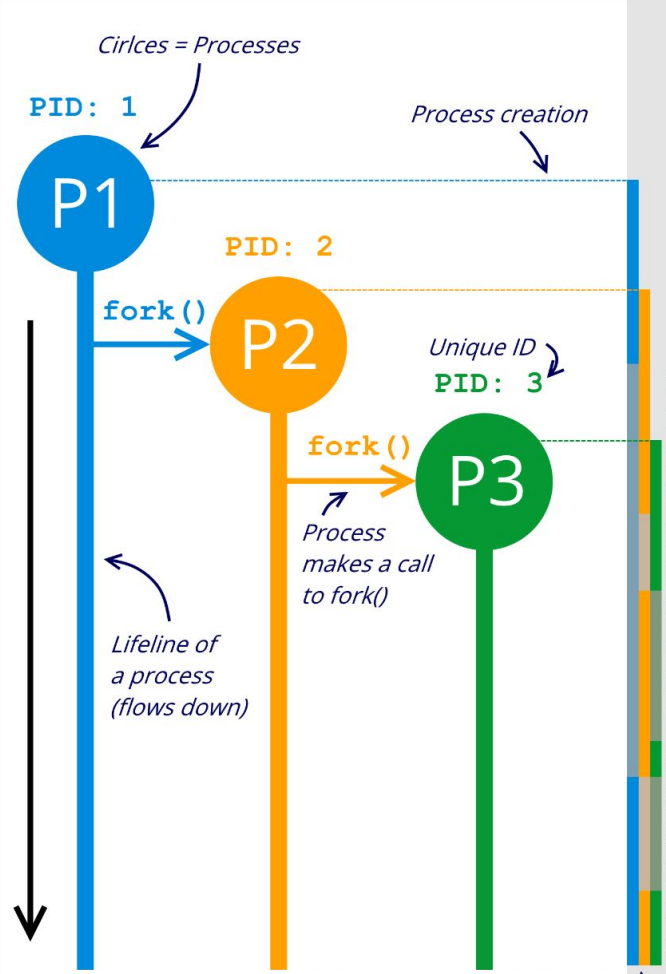
pid1	0
pid2	3

pid1	0
pid2	0

Which process printed “B”?

Output:

A B




```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5  int main() {
6      pid_t pid1, pid2;
7      pid1 = fork();
8
9      if (pid1 == 0) {
10         printf("A");
11         pid2 = fork();
12
13         if (pid2 == 0) {
14             printf("B");
15         }
16         else {
17             wait(NULL);
18             printf("C");
19         }
20     }
21     else {
22         printf("D");
23     }
24     return 0;
25 }
```

Lines of code touched per each process

SER 334
Processes

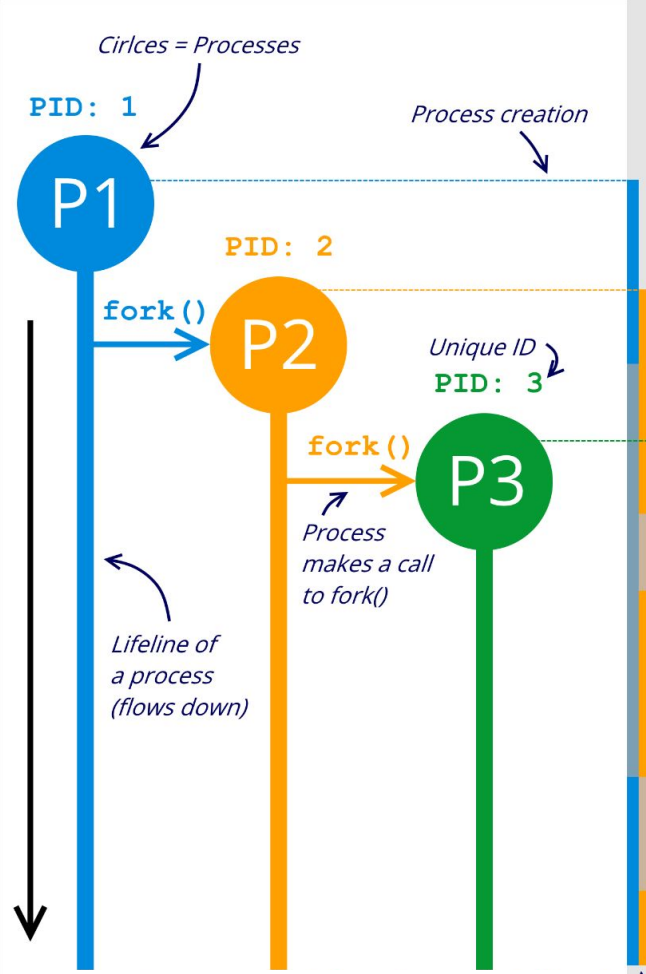
pid1	2
pid2	

pid1	0
pid2	3

pid1	0
pid2	0

Output:

A B




```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5  int main() {
6      pid_t pid1, pid2;
7      pid1 = fork();
8
9      if (pid1 == 0) {
10         printf("A");
11         pid2 = fork();
12
13         if (pid2 == 0) {
14             printf("B");
15         }
16         else {
17             wait(NULL);
18             printf("C");
19         }
20     }
21     else {
22         printf("D");
23     }
24     return 0;
25 }
```

?

Lines of code touched per each process

SER 334
Processes

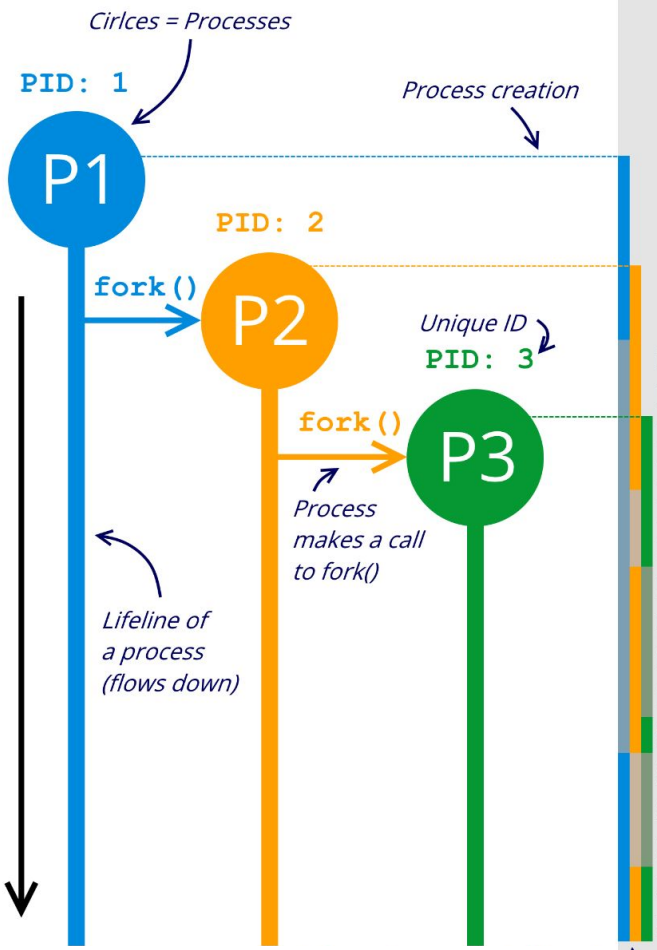
pid1	2
pid2	

pid1	0
pid2	3

pid1	0
pid2	0

Output:

A B C




```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5  int main() {
6      pid_t pid1, pid2;
7      pid1 = fork();
8
9      if (pid1 == 0) {
10         printf("A");
11         pid2 = fork();
12
13         if (pid2 == 0) {
14             printf("B");
15         }
16         else {
17             wait(NULL);
18             printf("C");
19         }
20     }
21     else {
22         printf("D");
23     }
24     return 0;
25 }
```

Lines of code touched per each process

SER 334
Processes

pid1	2
pid2	

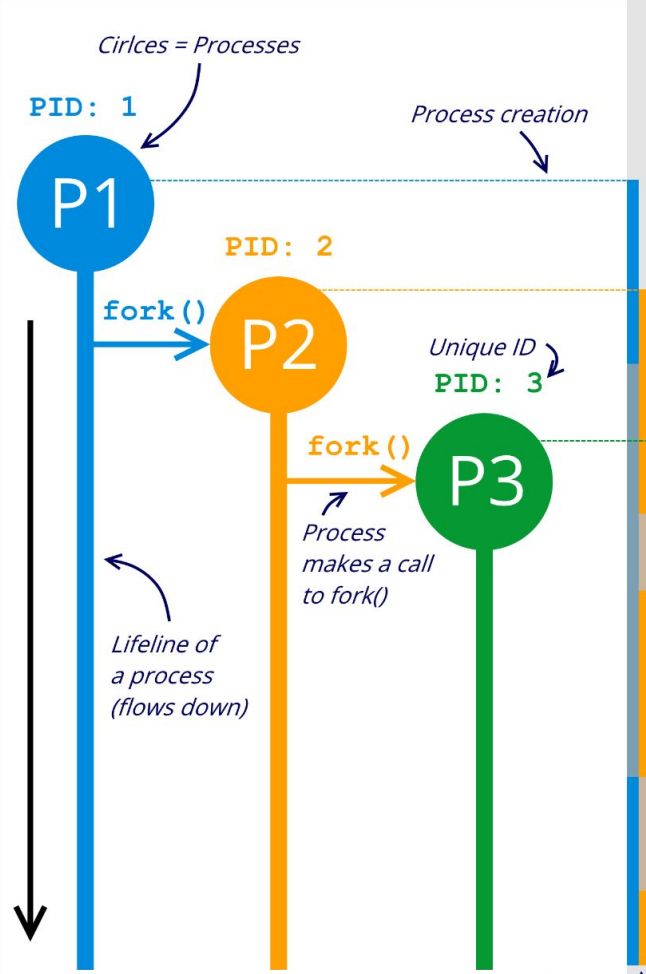
pid1	0
pid2	3

pid1	0
pid2	0

Which process printed “D”?

Output:

A B C D




```
1  #include <sys/types.h>
2  #include <stdio.h>
3  #include <unistd.h>
4
5  int main() {
6      pid_t pid1, pid2;
7      pid1 = fork();
8
9      if (pid1 == 0) {
10         printf("A");
11         pid2 = fork();
12
13         if (pid2 == 0) {
14             printf("B");
15         }
16         else {
17             wait(NULL);
18             printf("C");
19         }
20     }
21     else {
22         printf("D");
23     }
24     return 0;
25 }
```

Lines of code touched per each process

SER 334
Processes

pid1	2
pid2	

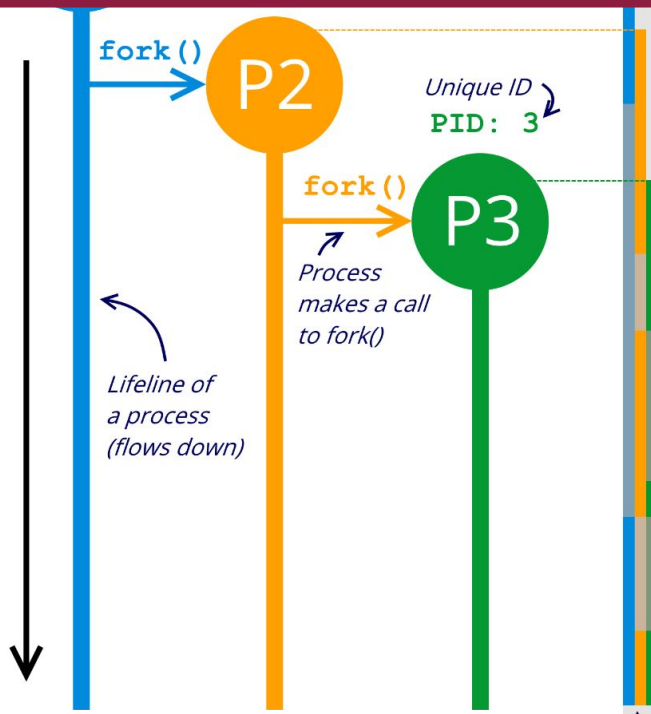
pid1	0
pid2	3

pid1	0
pid2	0

Output:

A B C D

Is that the only solution?




```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid1, pid2;
    pid1 = fork();

    if (pid1 == 0) {
        printf("A");
        pid2 = fork();

        if (pid2 == 0) {
            printf("B");
        }
        else {
            wait(NULL);
            printf("C");
        }
    }
    else {
        printf("D");
    }
    return 0;
}
```

Lines of code touched per each process

SER 334
Processes

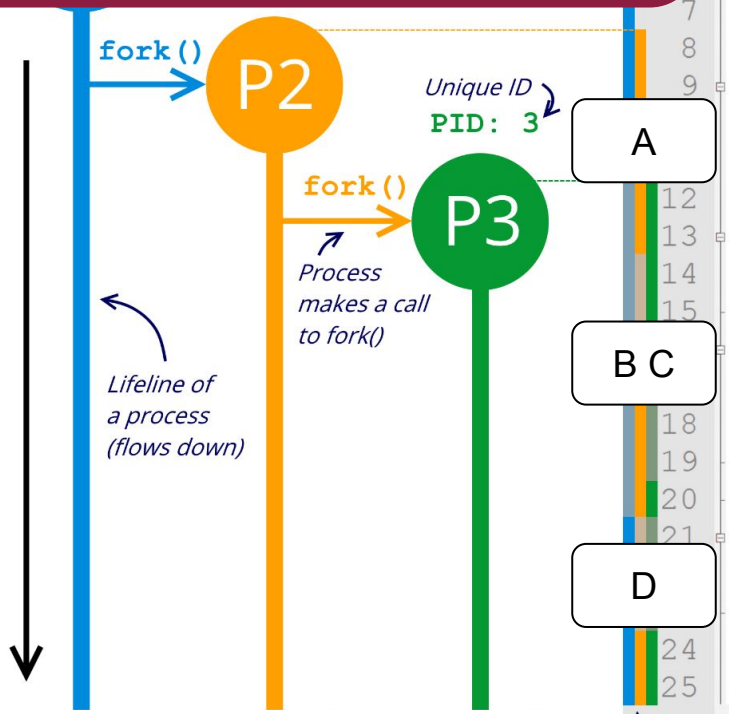
pid1	2
pid2	

pid1	0
pid2	3

pid1	0
pid2	0

Output:
A B C D

Is that the only solution?




```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid1, pid2;
    pid1 = fork();

    if (pid1 == 0) {
        printf("A");
        pid2 = fork();

        if (pid2 == 0) {
            printf("B");
        }
        else {
            wait(NULL);
            printf("C");
        }
    }
    else {
        printf("D");
    }
    return 0;
}
```

A

B C

D

Lines of code touched per each process

SER 334
Processes

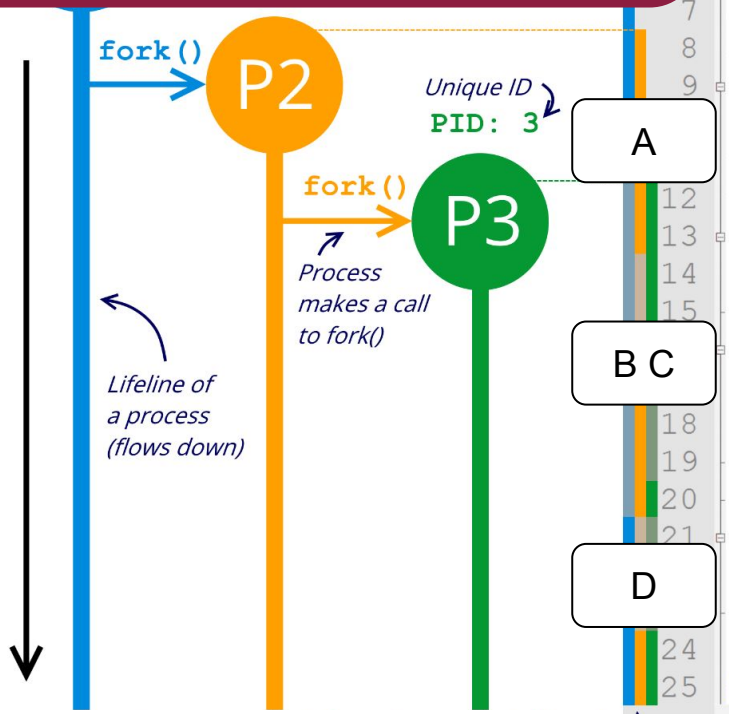
pid1	2
pid2	

pid1	0
pid2	3

pid1	0
pid2	0

Output:
A B C D
A D B C
D A B C
A B D C

Is that the only solution?




```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid1, pid2;
    pid1 = fork();

    if (pid1 == 0) {
        printf("A");
        pid2 = fork();

        if (pid2 == 0) {
            printf("B");
        }
        else {
            wait(NULL);
            printf("C");
        }
    }
    else {
        printf("D");
    }
    return 0;
}
```

A

B C

D

Lines of code touched per each process

Module 5 Sample

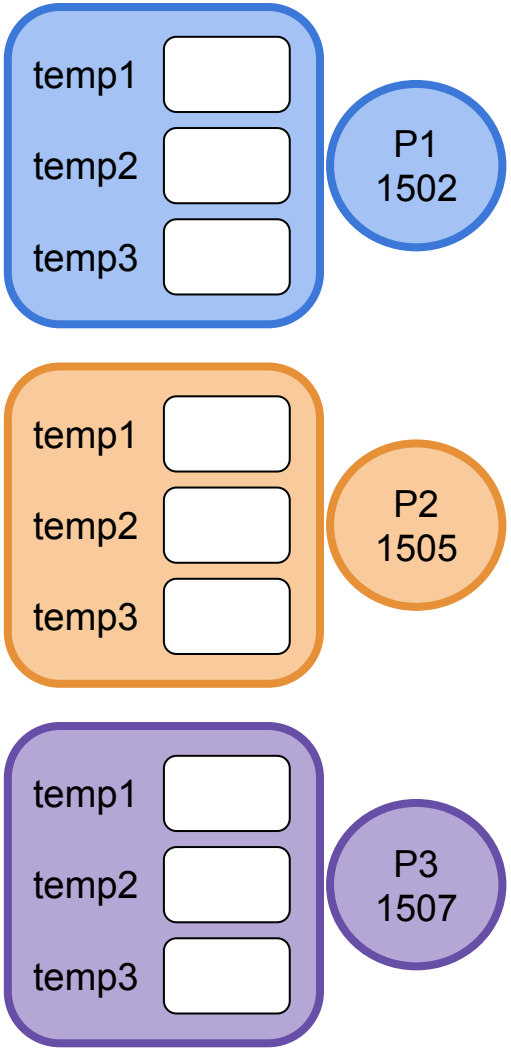
SER 334

Module 5 Sample

4. [Lisonbee] Trace the program below, identify the values of the pids at lines A, B, C, D, E, and F. (Assume that the actual pid of Process 1 is 1502, Process 2 is 1505, and Process 3 is 1507. Also assume that fork will always succeed.) [4 points]

A	
B	
C	
D	
E	
F	

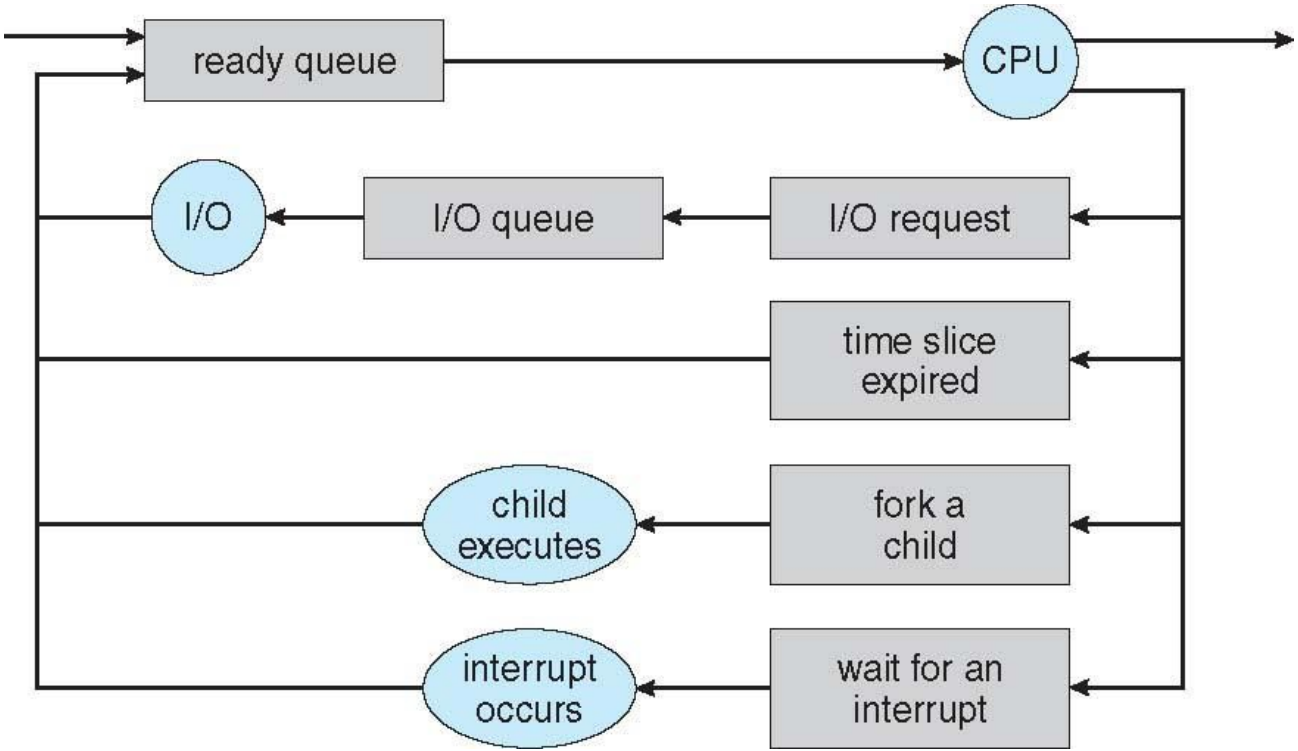
```
int main() {
    pid_t temp1, temp2, temp3;
    temp1 = fork();

    if (temp1 < 0) { /* Error occurred */
        fprintf(stderr, "Fork Failed");
        return 1;
    }
    else if (temp1 == 0) { /* Process 2 */
        temp2 = fork();

        if (temp2 < 0) { /* Error occurred */
            fprintf(stderr, "Fork Failed");
            return 1;
        }
        else if (temp2 == 0) { /* Process 3 */
            temp3 = getpid();
            printf("temp2 = %d", temp2); /* A */
            printf("temp3 = %d", temp3); /* B */
        }
        else { /* Process 2 */
            temp3 = getpid();
            printf("temp2 = %d", temp2); /* C */
            printf("temp3 = %d", temp3); /* D */
            wait(NULL);
        }
    }
    else { /* Process 1 */
        temp2 = getpid();
        printf("temp1 = %d", temp1); /* E */
        printf("temp2 = %d", temp2); /* F */
        wait(NULL);
    }
    return 0;
}
```



SER 334

Enter topic here

3. [Bahremand] Explain why the queue diagram in slide 9 has a continuous cycle that flows between the listed queues and resources? Is a cycle necessary? [2 points]



SER 334

Enter topic here

6. [Lisonbee] Consider a system where two processes (a producer and a consumer) use a message-passing system to communicate, and each process does work at different rates. The producer can produce (perform work) at any rate, but the consumer has to wait for the producer to complete its task before moving on. Based on this system's needs, explain whether a synchronous or asynchronous communication system would be a better choice and why. [2 points]

Synchronous

Asynchronous

SER 334

Scratch Space

Upcoming Events

SI Sessions:

- Thursday, February 1st at 7:00 pm MST
- Sunday, February 4th at 7:00 pm MST
- Monday, February 5th at 7:00 pm MST

Review Sessions:

- Exam 2 Review: TBD
- Exam 3 Review: TBD

Questions?

Survey:

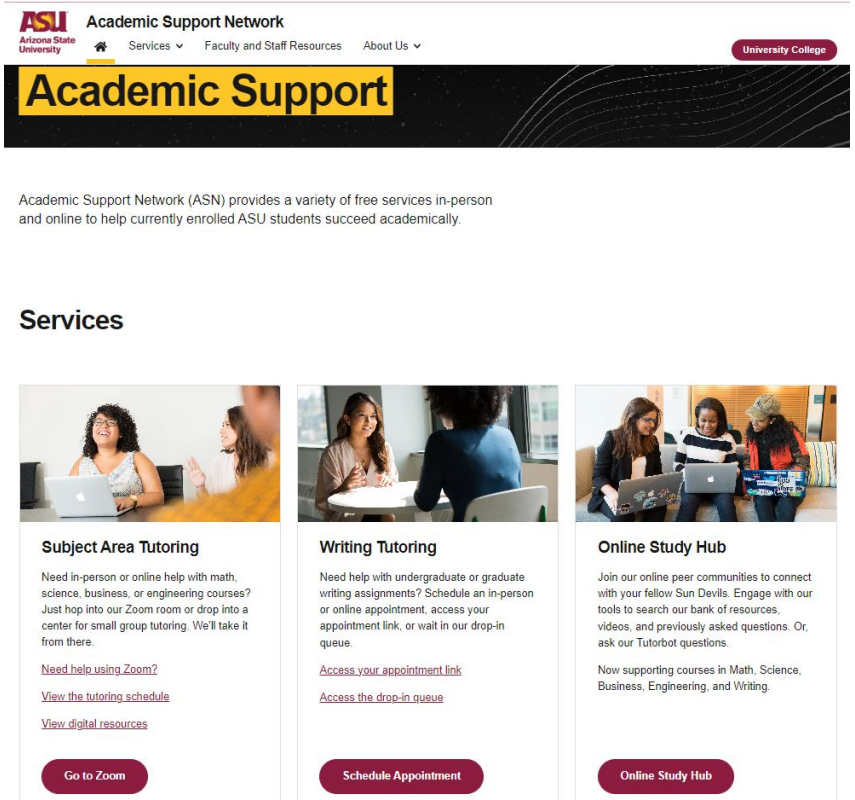
http://bit.ly/ASN2324

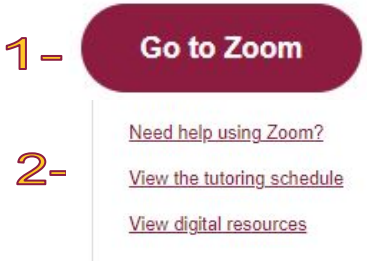
32

More Questions?

Check out our other resources!

tutoring.asu.edu



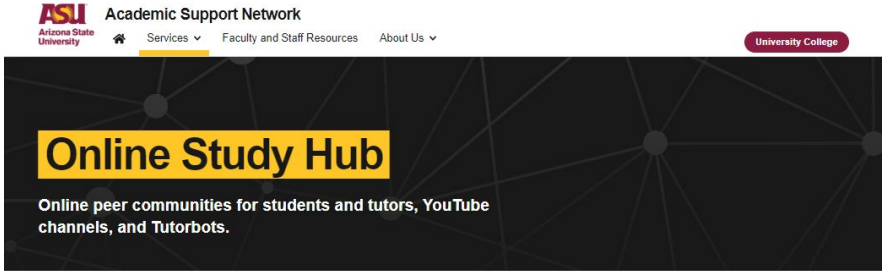


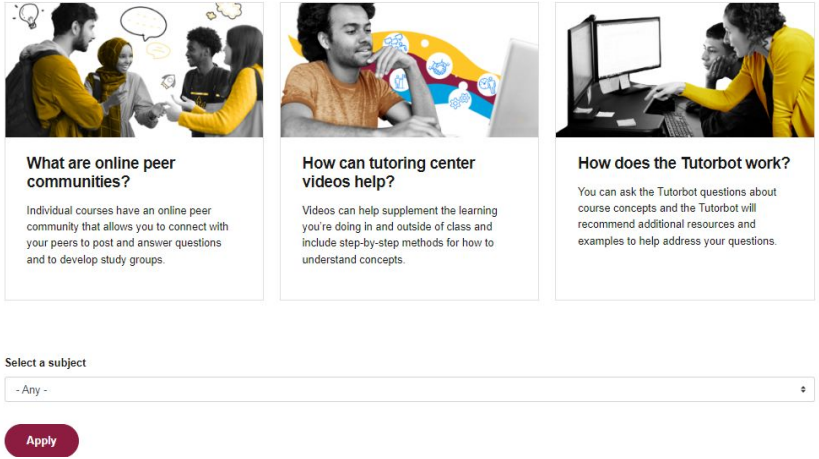
1. **Click on 'Go to Zoom' to log onto our Online Tutoring Center.**
2. **Click on 'View the tutoring schedule' to see when tutors are available for specific courses.**

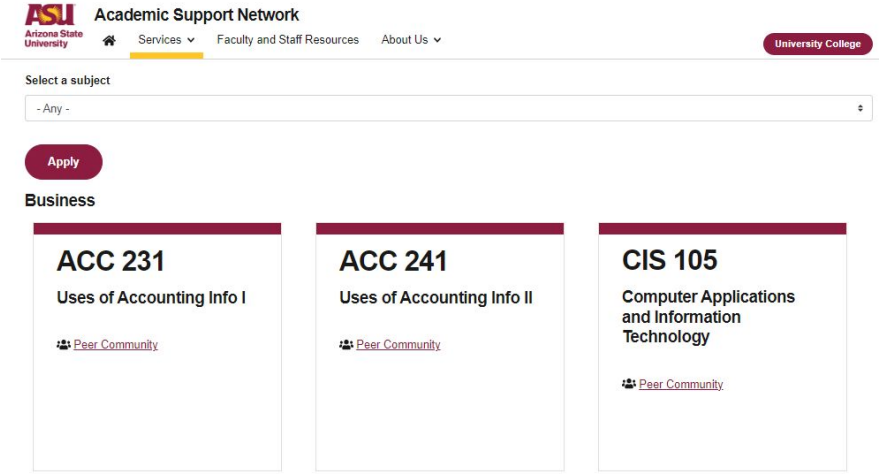
More Questions?

Check out our other resources!

tutoring.asu.edu/online-study-hub







Don't forget to check out the Online Study Hub for additional resources!

Additional Resources

- Course Repo
- Course Discord
- BMP File Format (Wiki)
- Linux Kernel API
- Bootlin - Linux Cross Referencer
- Dining Philosophers Interactive