Parallel and Concurrent Programming CS3S666

Delivery

- 24 Hours of Lectures
- 24 Hours of Tutorials
- 1 Coursework
- 1 Exam

Technology

- We will be using linux in (either vmare or raspberrypi) for the tutorials
- We will be using C++
- Lots of theory, and lots of technical content
  - All of which will be tested in the Exam, and the coursework

What is parallel programming?

Enter Text and Press Send

What is concurrent programming?

Enter Text and Press Send

Parallel Programming

- Getting lots of processors to complete one task

Concurrent Programming

- Getting one processor to complete several tasks

The difference is important

- Not all problems can be parallelized
- Sometimes parallelisation can make things worse (threads locking)

Concurrency

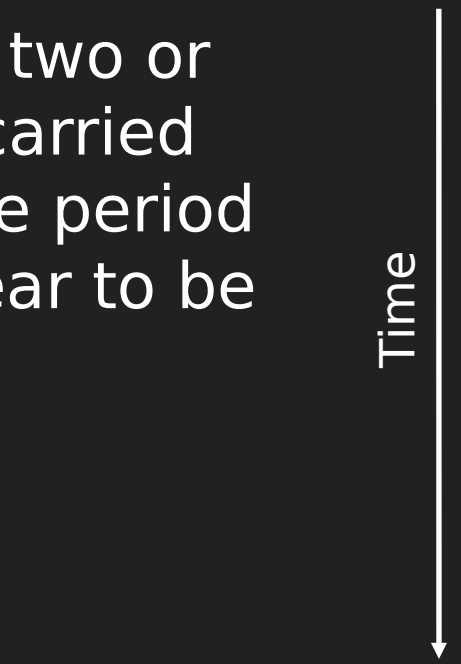
- To understand concurrency start with the opposite of concurrent:

Consecutive!

Task A - Start
Task A – Do Work
Task A – Finish
Task B – Start
Task B – Do Work
Task B – Finish
Task C – Start
Task C – Do

Concurrency

- Concurrency is that two or more tasks can be carried out in the same time period such that they appear to be overlapping.



Task A - Start
Task A – Do Work
Task B – Start
Task C – Start
Task B – Do Work
Task B – Finish
Task C – Do Work
Task C – Finish

A task could be a process, a thread, an asynchronous call etc. etc.

Concurrency

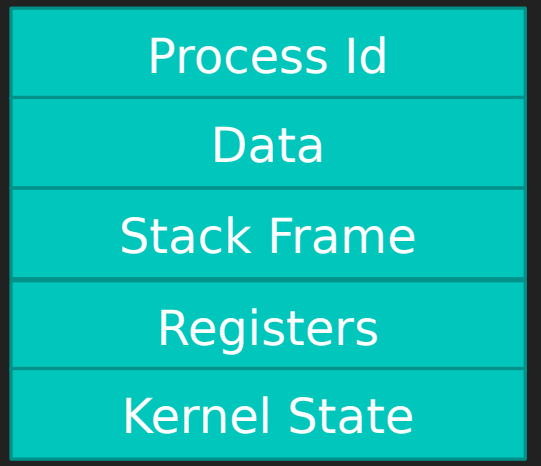
- The tasks are not happening physically at the same time:

Task A - Start	**Task B – Start**	**Task C – Start**
Task A – Do Work	**Task B – Do Work**	**Task C – Do Work**
Task A – Finish	**Task B – Finish**	**Task C – Finish**

- This is parallel programming, we will come onto this later

Processes

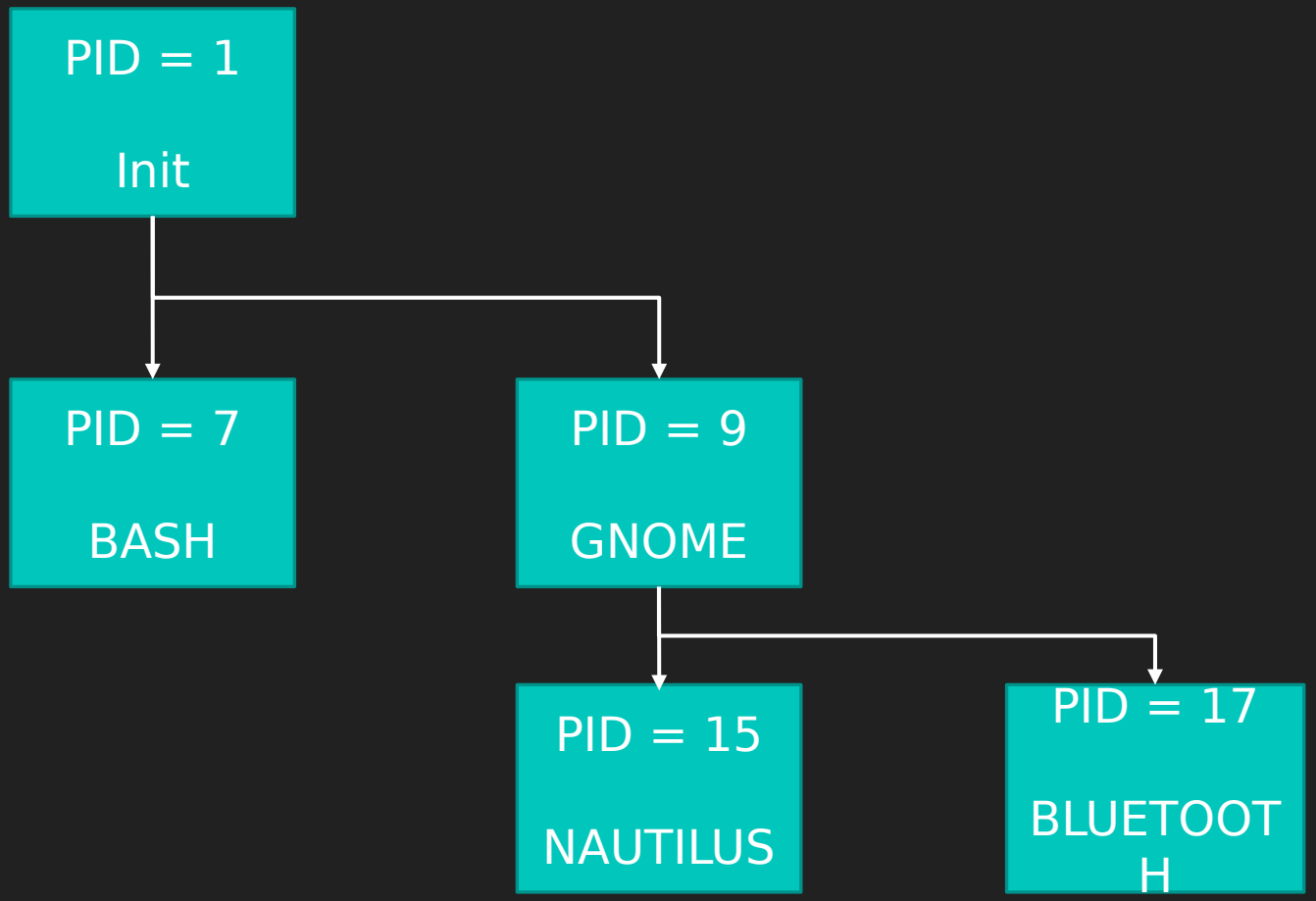
- A process is a collection of information used by the OS to keep track of a task.
- Task Manager



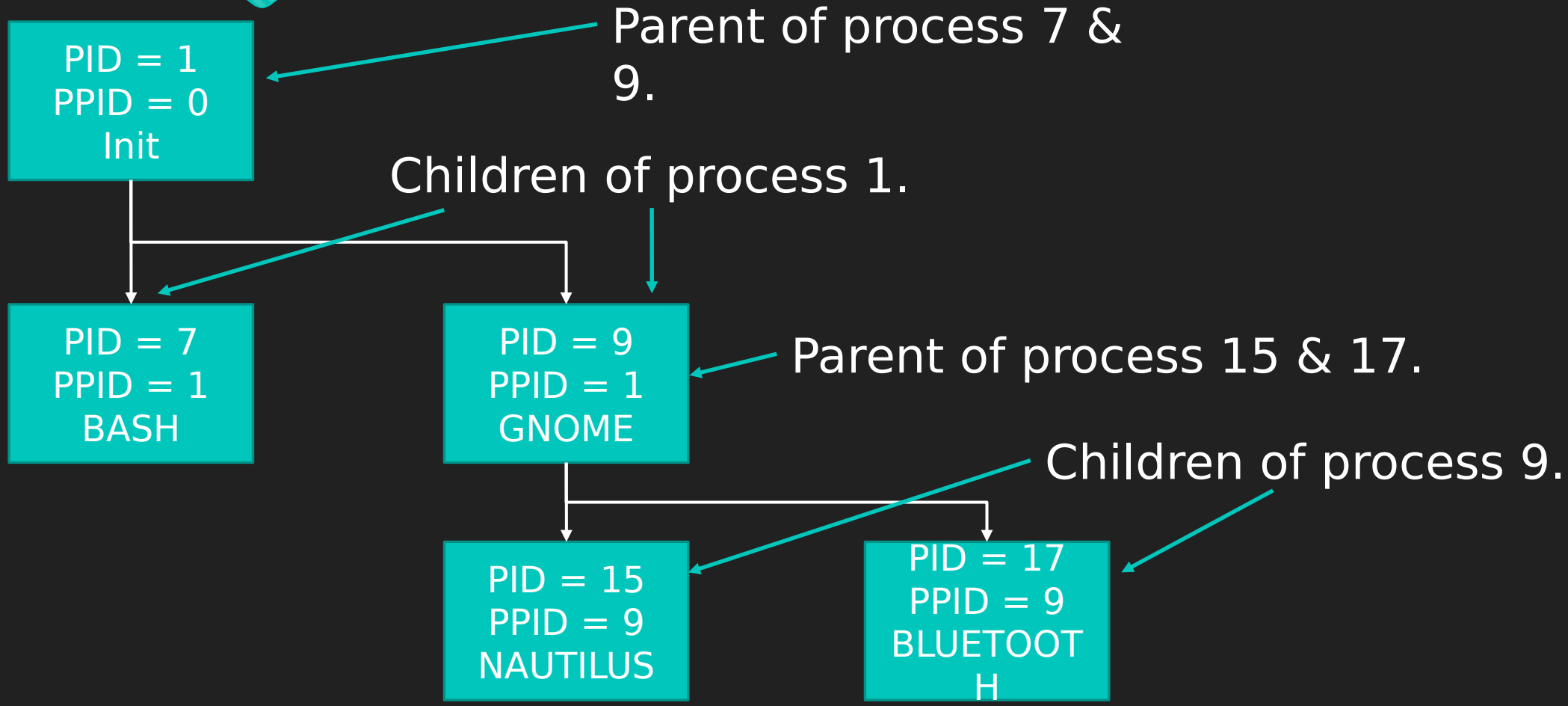
Process Hierarchy

- The Process ID (referred to as PID) is a unique number assigned to the process by the OS.
- In Unix a process will always exist with PID = 1, this is called the ***init*** process.
- A process can spawn (create) child processes.
  - E.g. using command window.

Process Hierarchy



Each Process knows Its parent PID (PPID).



Process Hierarchy

```
bash-4.4$ ps  xao pid,ppid,pgid,sid | head
  PID  PPID  PGID   SID
    1     0     1     1
   11     1    11    11
   17     1    17    17
   19    17    19    17
   21    19    21    17
   22    19    21    17
```

Creating a Process

- Processes are created using fork() command.
- This fork creates an exact duplicate of the process, except it has a different PID. Therefore it continues afterwards as if nothing happened!
- The fork command returns an integer:

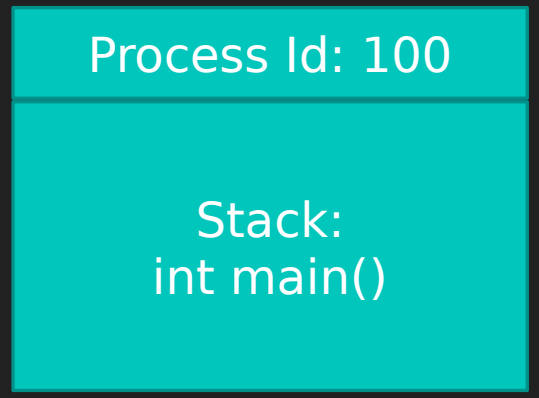
  0 if child process.

  -1 if fork call fails.

  Otherwise the child’s PID returned to parent process.

Creating a Process

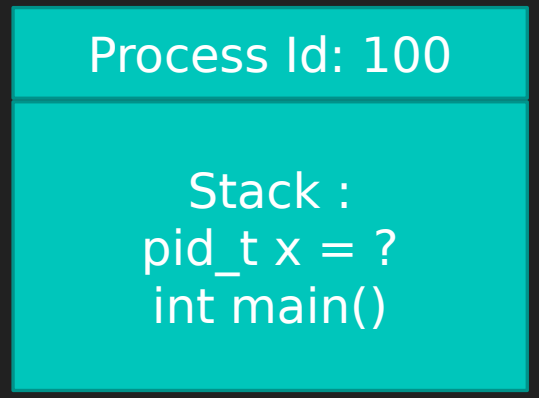
```
#include<iostream>
#include <unistd.h>

int main()
{
    pid_t x = fork();
    std::cout << "Returned val = " << x << std::endl;
    return 0;
}
```



Creating a Process

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

int main()
{
    pid_t x = fork();
    std::cout << "Returned val = " << x << std::endl;
    return 0;
}
```

Before call to fork()



Creating a Process

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

int main()
{
    pid_t x = fork();
    std::cout << "Returned val = " << x << std::endl;
    return 0;
}
```

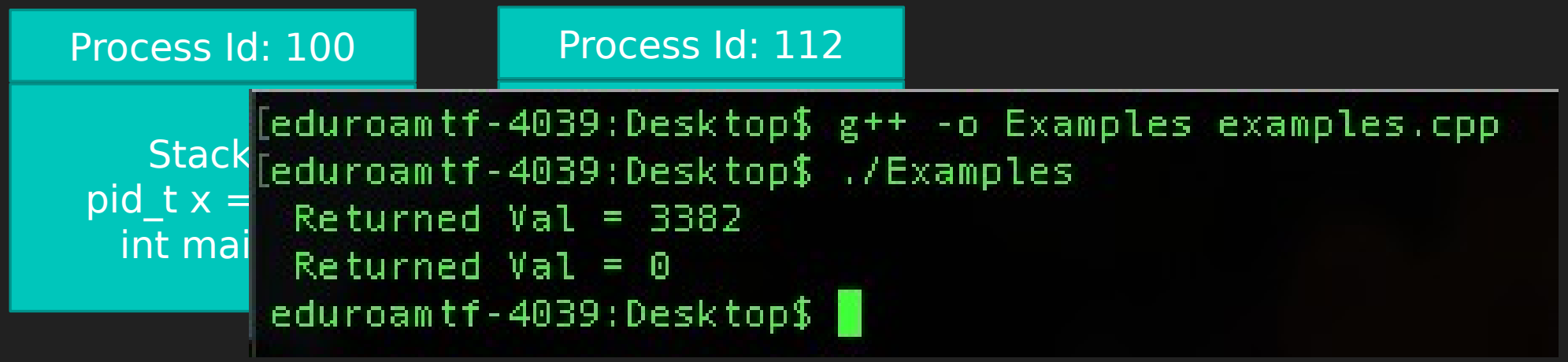
After call to fork()

Process Id: 100
Stack : pid_t x = 112 int main()

Process Id: 112
Stack : pid_t x = 0 int main()

Creating a Process

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

int main()
{
    pid_t x = fork();
    std::cout << "Returned Val = " << x << std::endl;
    return 0;
}
```



Testing the return

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

int main()
{
    pid_t x = fork();

    if (x == 0)
    {
        std::cout << "I am a child" << std::endl;
    }
    else
    {
        std::cout << "I am a parent" << std::endl;
    }
    return 0;
}
```

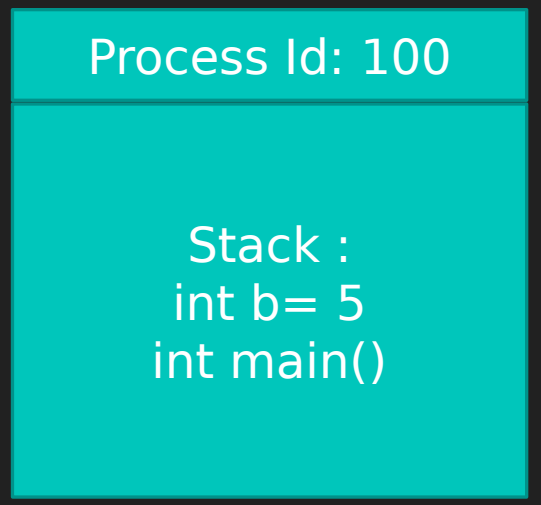
```
eduroamtf-4039:Desktop$ g++ -o Examples examples.cpp
eduroamtf-4039:Desktop$ ./Examples
i am a parent
I am a child
eduroamtf-4039:Desktop$
```

Creating a New Process

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

int main()
{
 int b = 5;
 pid_t x = fork();

 if (x == 0)
 {
    std::cout << "Child Process b = " << b << std::endl;
 }
 else
 {
    b = 10;
    std::cout << "Parent Process b = " << b << std::endl;
 }
 return 0;
}
```

What will be the output of this code?

Creating a New Process




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

int main()
{
 int b = 5;
 pid_t x = fork();

 if (x == 0)
 {
    std::cout << "Child Process b = " << b << std::endl;
 }
 else
 {
    b = 10;
    std::cout << "Parent Process b = " << b << std::endl;
 }
 return 0;
}
```

Creating a New Process

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

int main()
{
 int b = 5;
 pid_t x = fork();

 if (x == 0)
 {
    std::cout << "Child Process b = " << b << std::endl;
 }
 else
 {
    b = 10;
    std::cout << "Parent Process b = " << b << std::endl;
 }
 return 0;
}
```

Process Id: 100	Process Id: 120
Stack : pid_t x = 120 int b= 5 int main()	Stack : pid_t x = 0 int b= 5 int main()

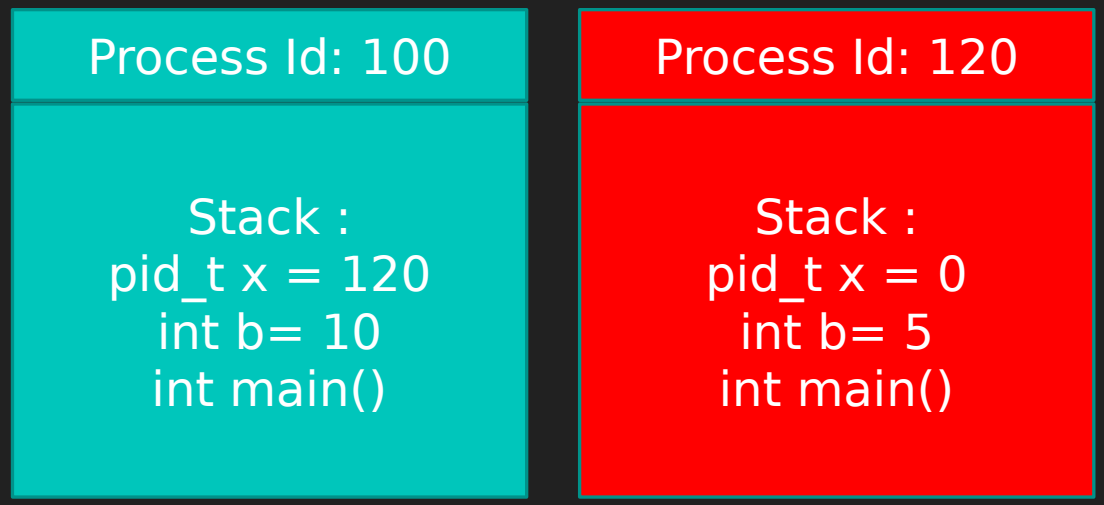
Creating a New Process

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

int main()
{
 int b = 5;
 pid_t x = fork();

 if (x == 0)
 {
    std::cout << "Child Process b = " << b << std::endl;
 }
 else
 {
    b = 10;
    std::cout << "Parent Process b = " << b << std::endl;
 }
 return 0;
}
```

Process Id: 100	Process Id: 120
Stack : pid_t x = 120 int b= 5 int main()	Stack : pid_t x = 0 int b= 5 int main()

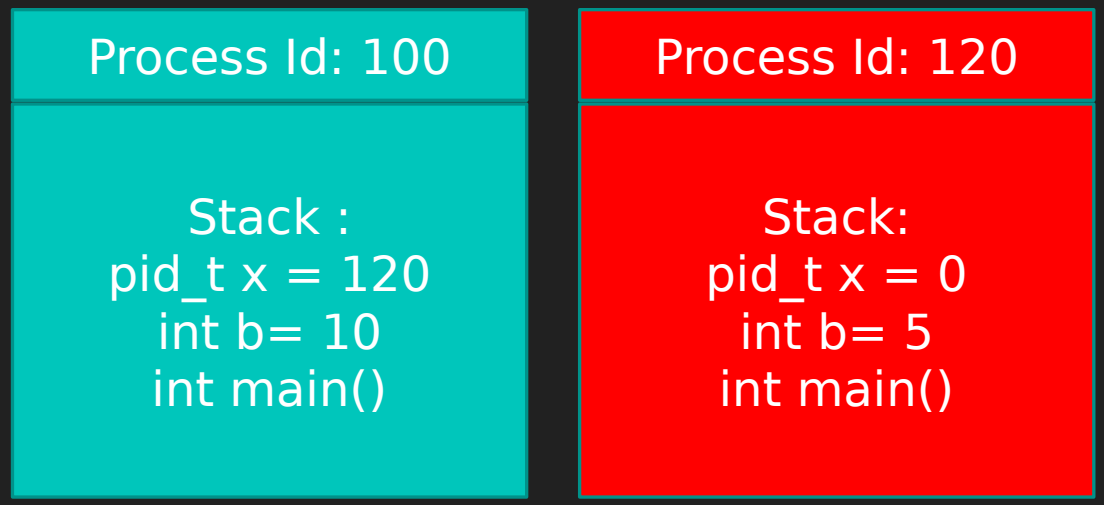
Creating a New Process

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

int main()
{
 int b = 5;
 pid_t x = fork();

 if (x == 0)
 {
    std::cout << "Child Process b = " << b << std::endl;
 }
 else
 {
    b = 10;
    std::cout << "Parent Process b = " << b << std::endl;
 }
 return 0;
}
```



Creating a New Process

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

int main()
{
 int b = 5;
 pid_t x = fork();

 if (x == 0)
 {
    std::cout << "Child Process b = " << b << std::endl;
 }
 else
 {
    b = 10;
    std::cout << "Parent Process b = " << b << std::endl;
 }
 return 0;
}
```




```
[eduroamtf-4039:Desktop$ g++ -o Examples examples.cpp
[eduroamtf-4039:Desktop$ ./Examples
Parent process b = 10
Child process b = 5
eduroamtf-4039:Desktop$
```

Memory

- What happens to the memory created on the heap?

Heap Allocated Memory

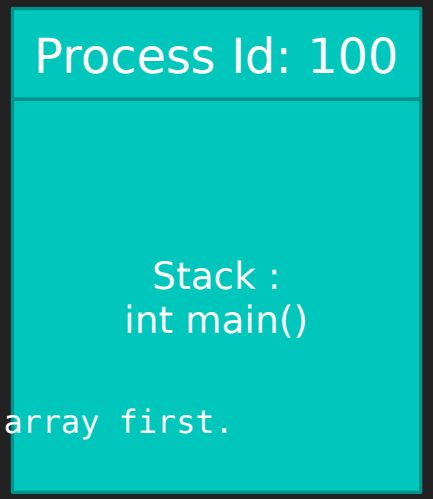
```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies array first.

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNu
     }

     delete[] pNums;
     return 0;
}
```

What will the output be now?

```
[eduroamtf-4039:Desktop$ g++ -o Examples examples.cpp
[eduroamtf-4039:Desktop$ ./Examples
Parent Process = 0
Parent Process = 1
Parent Process = 2
Parent Process = 3
Parent Process = 4
eduroamtf-4039:Desktop$ Child Process = 10
Child Process = 10
Child Process = 10
Child Process = 10
Child Process = 10
```

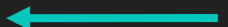
Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies array first.

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::endl;
     }

     delete[] pNums;
     return 0;
}
```



Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies a

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::endl;
     }

     delete[] pNums;
     return 0;
}
```



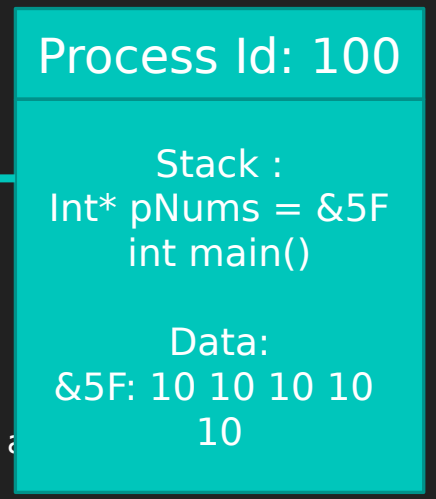
Process Id: 100

Stack :
Int* pNums = &5F
int main()

Data:
&5F: ? ? ? ? ?

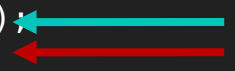
Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies a

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::endl;
     }

     delete[] pNums;
     return 0;
}
```



Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies a

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::endl;
     }

     delete[] pNums;
     return 0;
}
```



Process Id: 100	Process Id: 110
Stack : Pid_t = 110 Int* pNums = &5F int main() Data: &5F: 10 10 10 10 10	Stack : Pid_t = 0 Int* pNums = &5F int main() Data: &5F: 10 10 10 10 10

Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies a

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::endl;
     }

     delete[] pNums;
     return 0;
}
```

Process Id: 100	Process Id: 110
Stack : Pid_t = 110 Int* pNums = &5F int main() Data: &5F: 0 1 2 3 4	Stack : Pid_t = 0 Int* pNums = &5F int main() Data: &5F: 10 10 10 10 10

Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies a

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::endl;
     }

     delete[] pNums;
     return 0;
}
```

Process Id: 100	Process Id: 110
Stack : Pid_t = 110 Int* pNums = &5F int main() Data: &5F: 0 1 2 3 4	Stack : Pid_t = 0 Int* pNums = &5F int main() Data: &5F: 10 10 10 10 10

```
eduroamtf-4039:Desktop$ g++ -o Examples examples.cpp
eduroamtf-4039:Desktop$ ./Examples
Parent Process = 0
Parent Process = 1
Parent Process = 2
Parent Process = 3
Parent Process = 4
```

Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies a

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::endl
     }

     delete[] pNums;
     return 0;
}
```

Process Id: 100	Process Id: 110
Stack : Pid_t = 110 Int* pNums = &5F int main() Data: &5F:	Stack : Pid_t = 0 Int* pNums = &5F int main() Data: &5F: 10 10 10 10 10

```
eduroamtf-4039:Desktop$ g++ -o Examples examples.cpp
eduroamtf-4039:Desktop$ ./Examples
Parent Process = 0
Parent Process = 1
Parent Process = 2
Parent Process = 3
Parent Process = 4
```

Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies a

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::endl
     }

     delete[] pNums;
     return 0;
}
```

Process Id: 100	Process Id: 110
Stack : Pid_t = 110 Int* pNums = &5F int main() Data: &5F:	Stack : Pid_t = 0 Int* pNums = &5F int main() Data: &5F: 10 10 10 10 10

```
eduroamtf-4039:Desktop$ g++ -o Examples examples.cpp
eduroamtf-4039:Desktop$ ./Examples
Parent Process = 0
Parent Process = 1
Parent Process = 2
Parent Process = 3
Parent Process = 4
```

Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies a

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::en
     }

     delete[] pNums;
     return 0;
}
```

Process Id: 100	Process Id: 110
Stack : Pid_t = 110 Int* pNums = &5F int main() Data: &5F:	Stack : Pid_t = 0 Int* pNums = &5F int main() Data: &5F: 10 10 10 10 10

```
eduroamtf-4039:Desktop$ g++ -o Examples examples.cpp
eduroamtf-4039:Desktop$ ./Examples
Parent Process = 0
Parent Process = 1
Parent Process = 2
Parent Process = 3
Parent Process = 4
eduroamtf-4039:Desktop$ Child Process = 10
Child Process = 10
Child Process = 10
Child Process = 10
Child Process = 10
```

Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies a

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::en
     }

     delete[] pNums;
     return 0;
}
```

Process Id: 100	Process Id: 110
Stack : Pid_t = 110 Int* pNums = &5F int main() Data: &5F:	Stack : Pid_t = 0 Int* pNums = &5F int main() Data: &5F:

```
eduroamtf-4039:Desktop$ g++ -o Examples examples.cpp
eduroamtf-4039:Desktop$ ./Examples
Parent Process = 0
Parent Process = 1
Parent Process = 2
Parent Process = 3
Parent Process = 4
eduroamtf-4039:Desktop$ Child Process = 10
Child Process = 10
Child Process = 10
Child Process = 10
Child Process = 10
```

Heap Allocated Memory

```
int main()
{
     int *pNums = new int[5];
     for (int i = 0; i<5; i++) pNums[i] = 10;

     pid_t x = fork();


     if(x == 0)
     {
          sleep(1); //Ensure parent process modifies a

          for (int i = 0; i<5; i++)
               std::cout << "Child Process = " << pNums[i] << std::endl;
     }
     else
     {
      //Change values stored in array.
          for (int i = 0; i<5; i++) pNums[i] = i;

          for (int i = 0; i<5; i++)
           std::cout << "Parent Process = " << pNums[i] << std::en
     }

     delete[] pNums;
     return 0;
}
```

Process Id: 100	Process Id: 110
Stack : Pid_t = 110 Int* pNums = &5F int main() Data: &5F:	Stack : Pid_t = 0 Int* pNums = &5F int main() Data: &5F:

```
eduroamtf-4039:Desktop$ g++ -o Examples examples.cpp
eduroamtf-4039:Desktop$ ./Examples
Parent Process = 0
Parent Process = 1
Parent Process = 2
Parent Process = 3
Parent Process = 4
eduroamtf-4039:Desktop$ Child Process = 10
Child Process = 10
Child Process = 10
Child Process = 10
Child Process = 10
```

What ?

```
int main()
{
     int *pNums = new int[5];
     pid_t x = fork();

     if (x == 0)
     {
          std::cout << "Child Memory Location: " << pNums <<
std::endl;
     }
     else
     {
          std::cout << "Parent Memory Location: " << pNums <<
std::endl;
     }

     delete[] pNums;
     return 0;
}
```

They have same value??

```
[eduroamtf-4039:Desktop$ g++ -o Examples examples.cpp
[eduroamtf-4039:Desktop$ ./Examples
 Parent Memory Location: 0x7fa574c00540
 Child Memory Location: 0x7fa574c00640
```

Heap Allocated Memory

- We know that heap allocated memory is different for each process (i.e. not shared).
- However, the pointer address has the same value for each process.
- How can this be?

Heap Allocated Memory

- The memory used by a process is “virtual”, it does not represent the physical location on the RAM stick.

- Effectively, the entire heap is copied during a fork()*.

* This is a gross simplification of the true process!!

Summary

- We have learnt about processes in Linux.
- We understand how to use fork() to duplicate a process.
- We understand the return values of fork().
- We understand how process memory is treated.

Next Time

- Exec() and pipes
- This weeks tutorial is about refreshing linux and c++ skills, and using the fork() command.

Reference

- Seven Concurrency Models in Seven Weeks, When Thread Unravel
  - Paul Butcher 2014
  - Available in the library