



# CS 1550

Week 5 - Project 2 Discussion

Teaching Assistant

Henrique Potter

# CS 1550 – Project 2 is out

---

- We will be using Qemu again for Project 2

# CS 1550 – Project 2 is out

---

- We will be using Qemu again for Project 2
- You will need a working semaphore implementation
  - You can reuse your Project 1 Linux kernel (bzImage)
  - Or you can download a working kernel from canvas.

# CS 1550 – Project 2 is out

---

- We will be using Qemu again for Project 2
- You will need a working semaphore implementation
  - You can reuse your Project 1 Linux kernel (bzImage)
  - Or you can download a working kernel from canvas.
- You only need to cp the working kernel once in Linux-devel
- Afterwards you will only need to compile your program in thoth and copy to Qemu (with SCP)

# Project 2 - Safe Museum Tour Problem

---

- A visitor can not tour a museum without a tour guide

# Project 2 - Safe Museum Tour Problem

---

- A visitor can not tour a museum without a tour guide



Visitor



Tour Guide

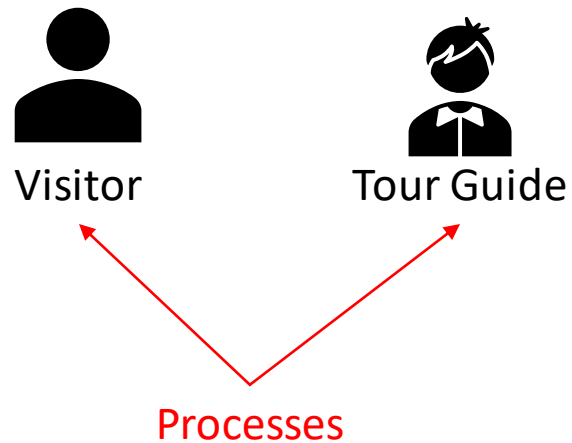
Museum



# Project 2 - Safe Museum Tour Problem

---

- A visitor can not tour a museum without a tour guide



# Project 2 - Safe Museum Tour Problem

---

- A visitor can not tour a museum without a tour guide

Main:

```
int pid = fork();           //create child process (visitor arrival)
if(pid == 0){               //child process when pid == 0
    VisitorArrivalProcess   //visitor arrival process
}
```



# Project 2 - Safe Museum Tour Problem

---

- A visitor can not tour a museum without a tour guide

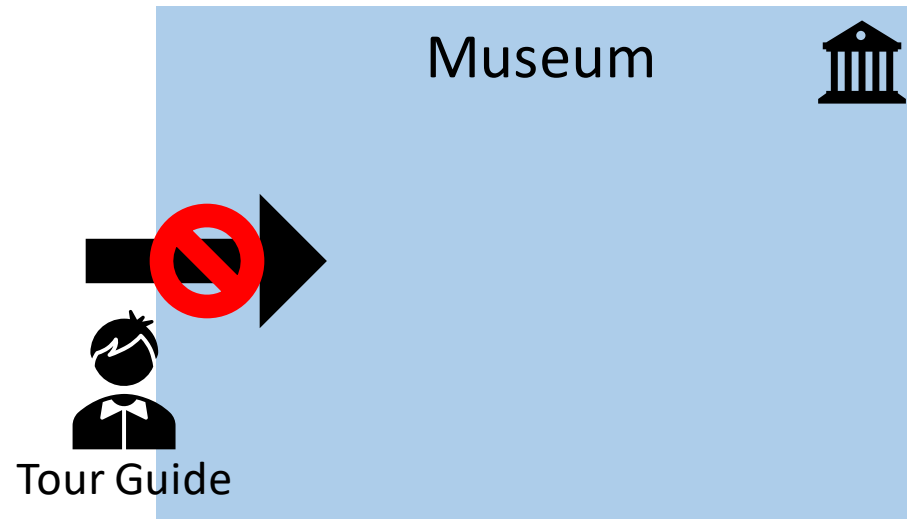
Main:

```
int pid = fork();           //create child process (visitor arrival)
if(pid == 0){               //child process when pid == 0
    VisitorArrivalProcess   //visitor arrival process
} else {                    //original process when pid != 0
    int pid = fork()        //create following child process(guides)
    if(pid == 0){           //child process when pid == 0
        Guides
    }
```

# Project 2 - Safe Museum Tour Problem

---

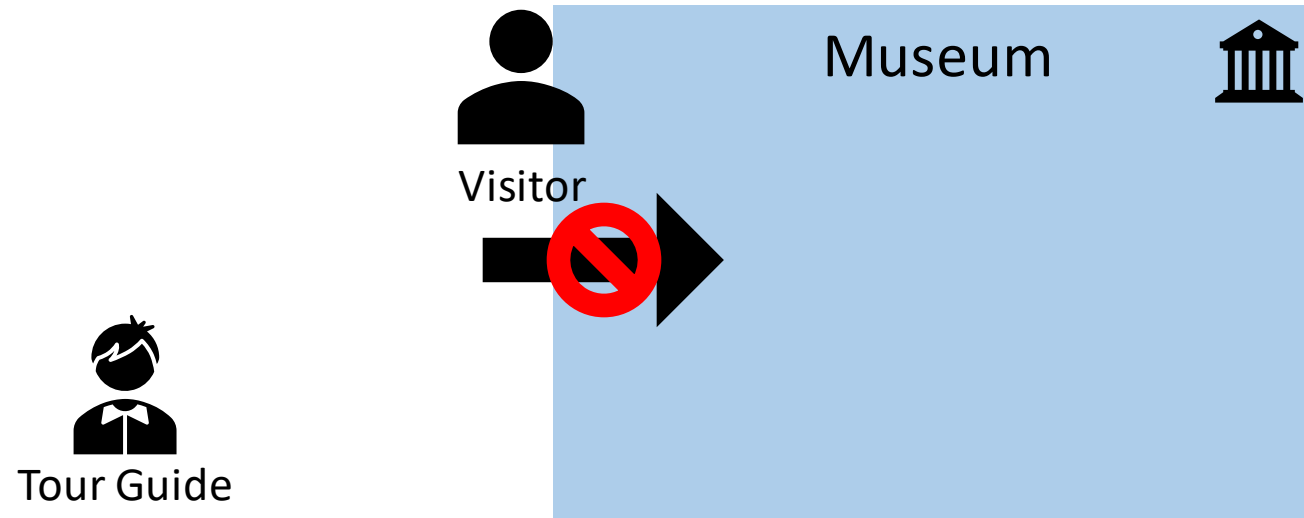
- A visitor can not tour a museum without a tour guide
- A tour guide cannot open the museum without a visitor



# Project 2 - Safe Museum Tour Problem

---

- A visitor can not tour a museum without a tour guide
- A tour guide cannot open the museum without a visitor



# Project 2 - Safe Museum Tour Problem

---

- A visitor can not tour a museum without a tour guide
- A tour guide cannot open the museum without a visitor

Main:

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    int pid = fork()
    if(pid == 0){
        Guides
    }
}
```

# Project 2 - Safe Museum Tour Problem

---

- A visitor can not tour a museum without a tour guide
- A tour guide cannot open the museum without a visitor

Main:

```
int pid = fork();  
if(pid == 0){  
    VisitorArrivalProcess  
} else {  
    int pid = fork()  
    if(pid == 0){  
        Guides  
    }  
}
```



Visitor



Tour Guide

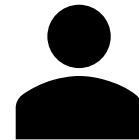
# Project 2 - Safe Museum Tour Problem

---

- A visitor can not tour a museum without a tour guide
- A tour guide cannot open the museum without a visitor

Main:

```
int pid = fork();  
if(pid == 0){  
    VisitorArrivalProcess  
} else {  
    int pid = fork()  
    if(pid == 0){  
        Guides  
    }  
}
```

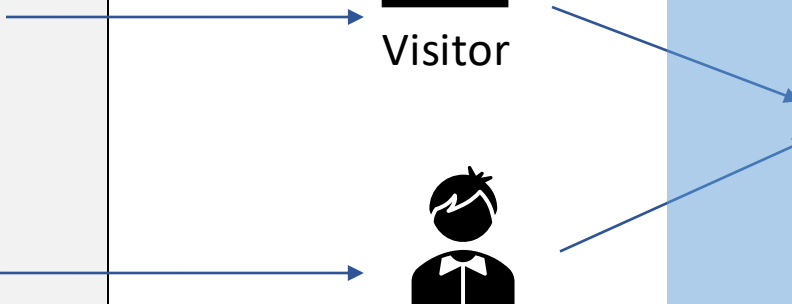


Visitor



Tour Guide

Museum



# Project 2 - Safe Museum Tour Problem

---

- A visitor can not tour a museum without a tour guide
- A tour guide cannot open the museum without a visitor
- A tour guide leaves when no more visitors are in the museum

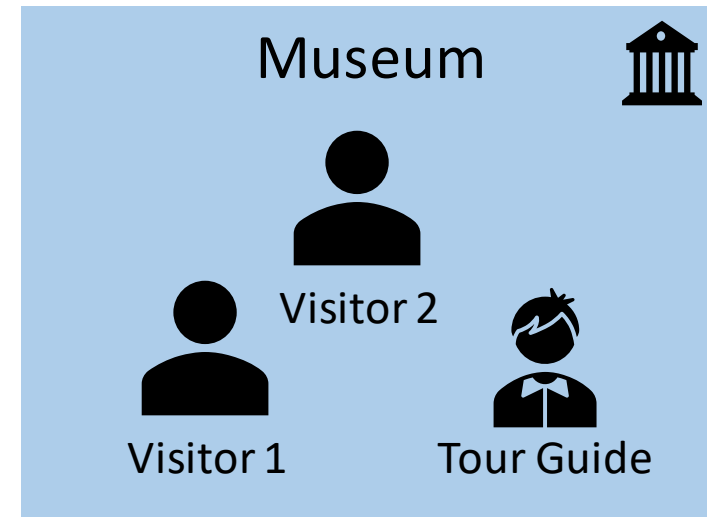
# Project 2 - Safe Museum Tour Problem

---

- A tour guide leaves when no more visitors are in the museum
- A tour guide **cannot leave until all visitors** in the museum leave

Main:

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    int pid = fork()
    if(pid == 0){
        Guides
    }
}
```





# Project 2 - Safe Museum Tour Problem

---

- A tour guide **leaves when no more visitors** are in the museum
- A tour guide cannot leave until all visitors in the museum leave

Main:

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    int pid = fork()
    if(pid == 0){
        Guides
    }
}
```

Museum



Tour Guide

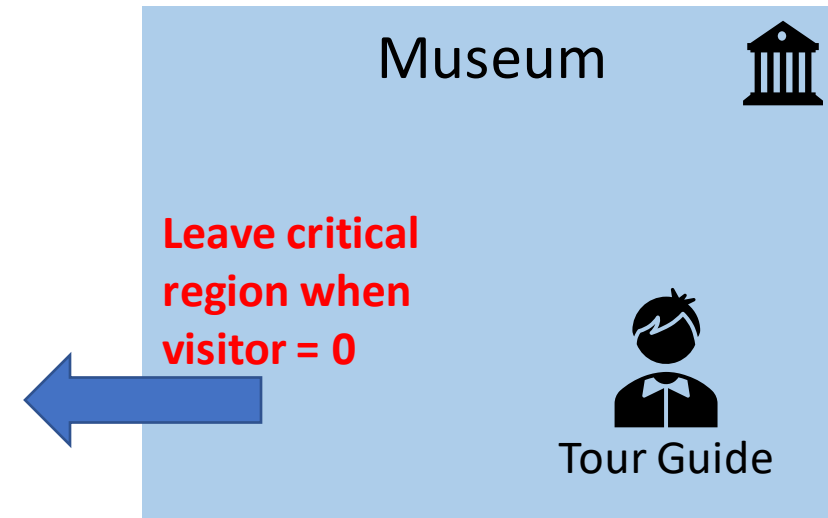
# Project 2 - Safe Museum Tour Problem

---

- A tour guide **leaves when no more visitors** are in the museum
- A tour guide cannot leave until all visitors in the museum leave

Main:

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    int pid = fork()
    if(pid == 0){
        Guides
    }
}
```



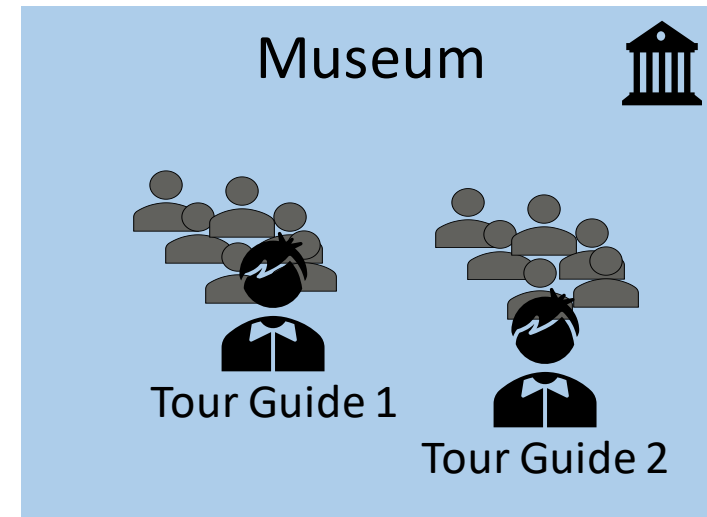
# Project 2 - Safe Museum Tour Problem

---

- At most **two tour guides** can be in the museum at a time
- Each tour guide provides a tour **for at most ten visitors**

Main:

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    int pid = fork()
    if(pid == 0){
        Guides
    }
}
```



# Project 2 - Safe Museum Tour Problem

---

- Your program **should always** satisfy those **constraints**
- Under **no conditions** will cause a **deadlock** to occur
  - When the museum is empty
  - Tour guide and a visitor arrive and wait outside forever

Main:

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    int pid = fork()
    if(pid == 0){
        Guides
    }
}
```

# Project 2 - Safe Museum Tour Problem

---

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)

# Project 2 - Safe Museum Tour Problem

---

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)

```
Main: visitorCount, tourguideCount
    int pid = fork();
    if(pid == 0){
        VisitorArrivalProcess
    } else {
        Guides
    }
```

You need to define specific behaviors to the creation of routines of both visitor and guides

50% to sleep after each creation

# Project 2 - Safe Museum Tour Problem

---

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival

```
Main: visitorCount, tourguideCount
```

```
    int pid = fork();
```

```
    if(pid == 0){
```

```
        VisitorArrivalProcess
```

```
    } else {
```

```
        Guides
```

```
    }
```

```
... VisitorArrivalProcess
```

```
for (i=0;i<visitorCount;i++){
```

```
    int pid = fork();
```

```
    if(pid == 0){
```

```
        visitor();
```

```
    }
```

```
}
```

# Project 2 - Safe Museum Tour Problem

---



```
Main: visitorCount, tourguideCount
    int pid = fork();
    if(pid == 0){
        VisitorArrivalProcess
    } else {
        Guides
    }
```



# Project 2 - Safe Museum Tour Problem

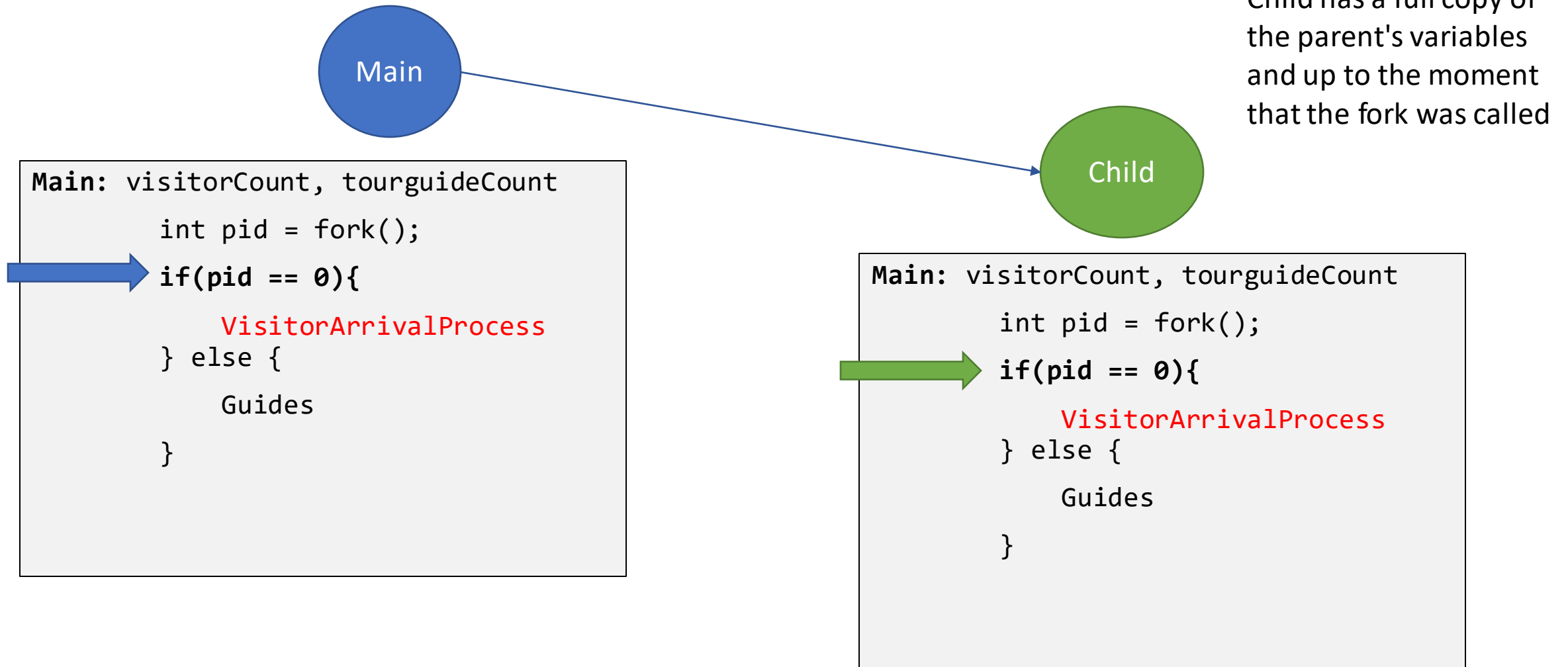
---



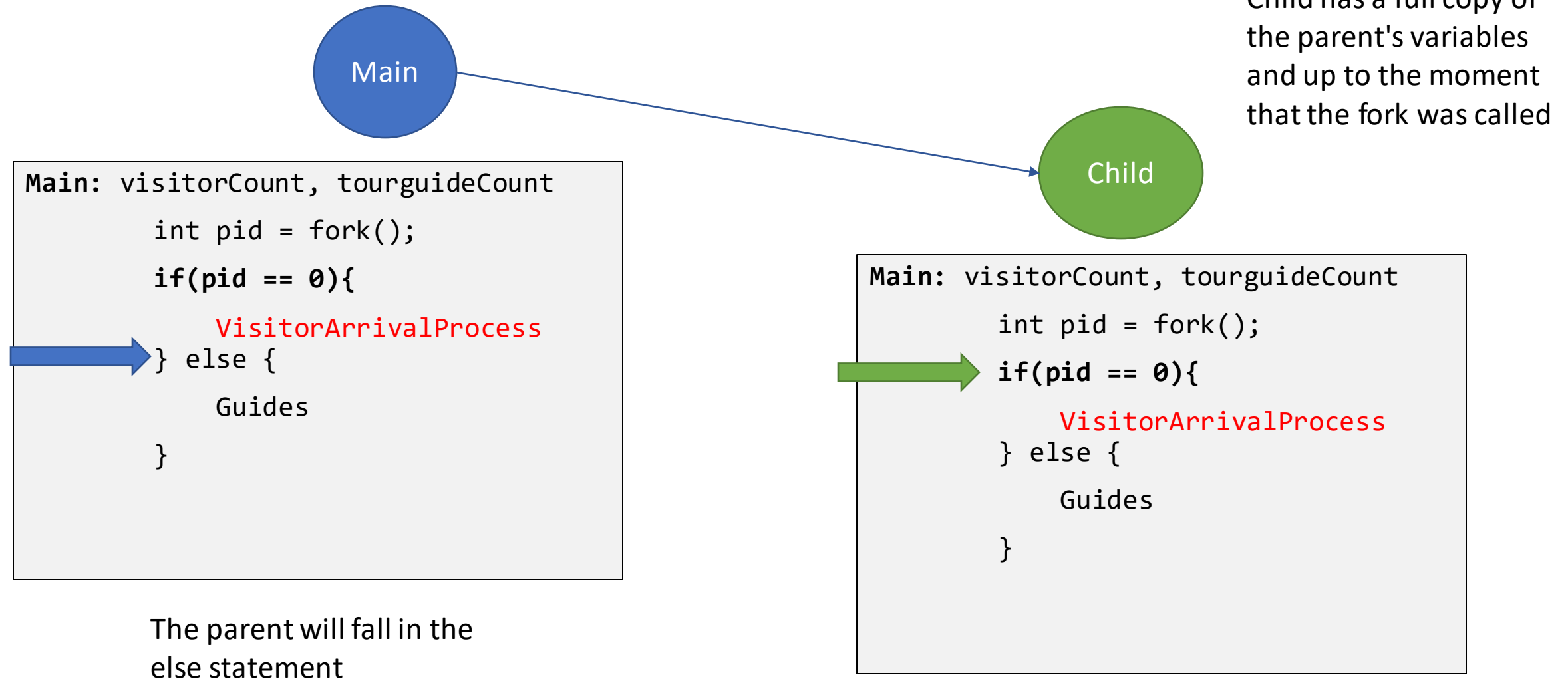
```
Main: visitorCount, tourguideCount
    int pid = fork();
    if(pid == 0){
        VisitorArrivalProcess
    } else {
        Guides
    }
```

Calling fork creates a new process. The **pid** is a value **bigger than 0** for the parent **but is 0 for the child**.

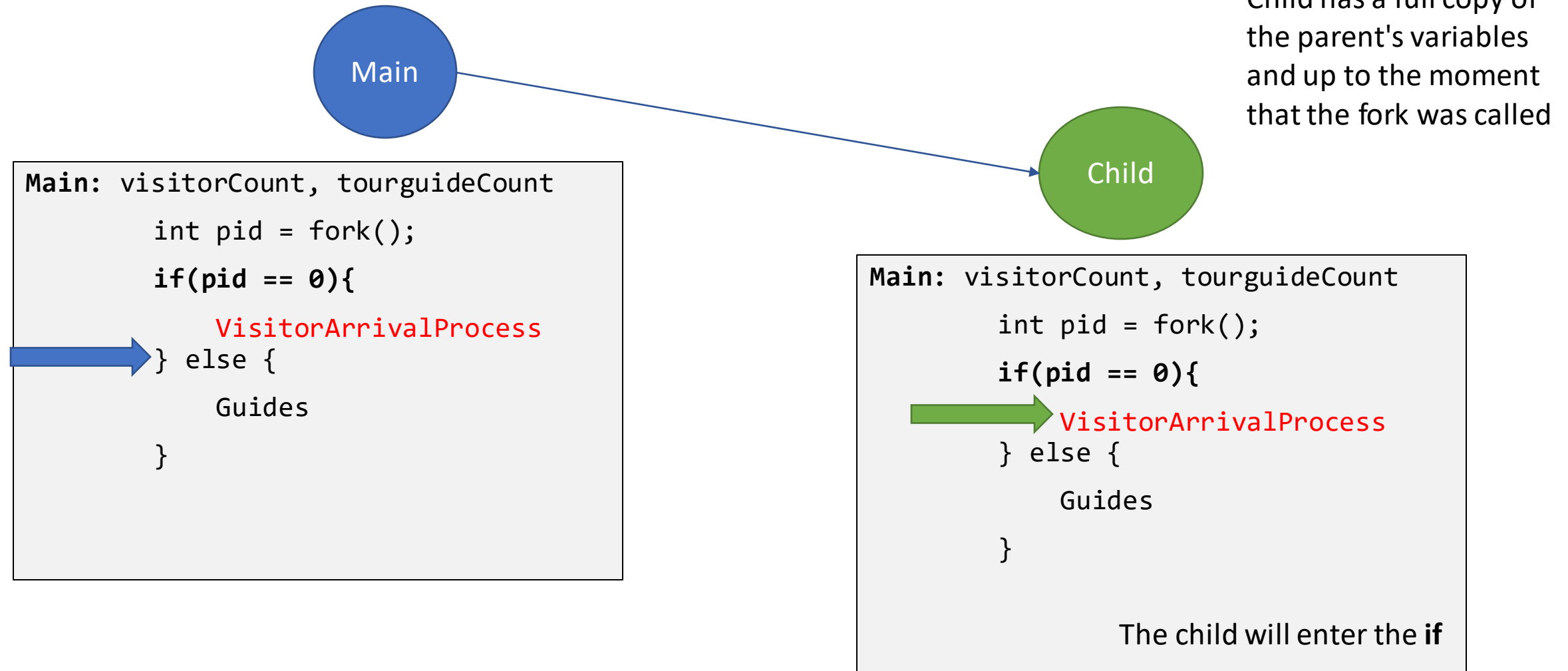
# Project 2 - Safe Museum Tour Problem



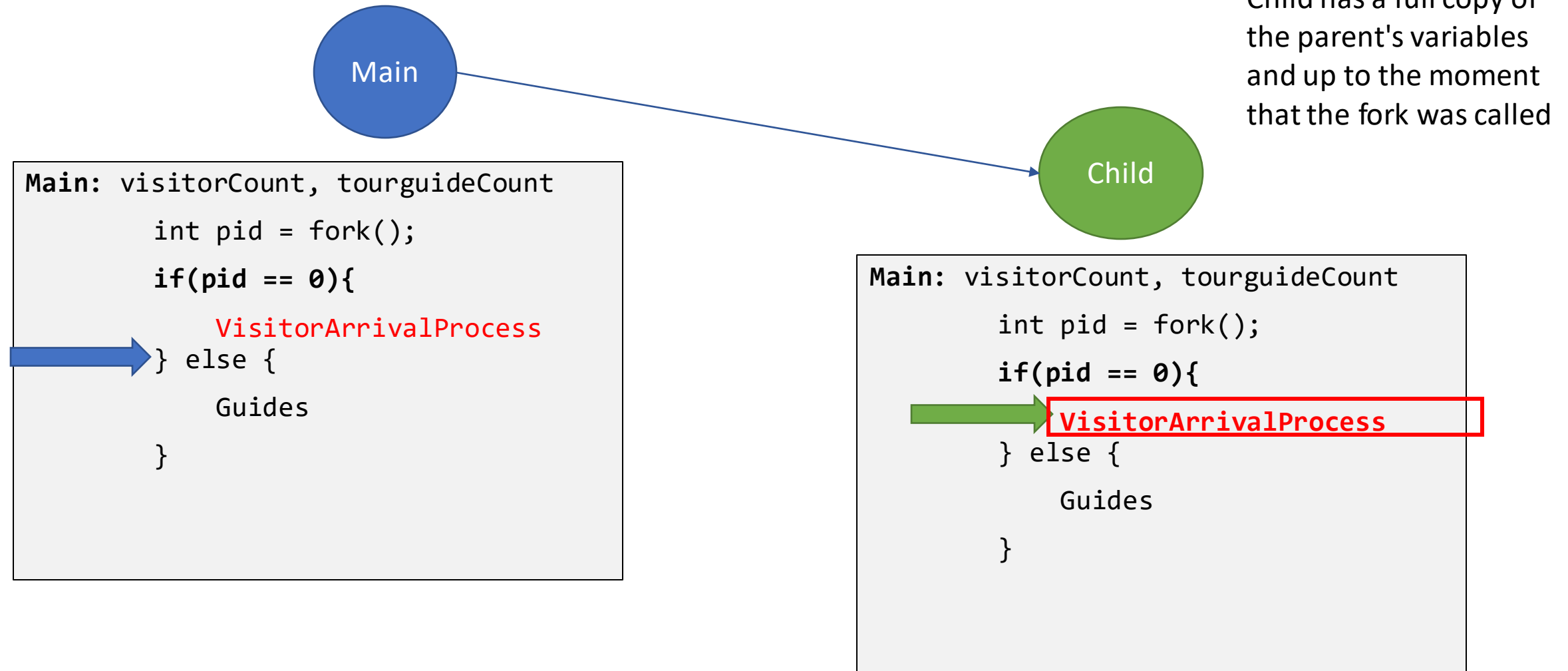
# Project 2 - Safe Museum Tour Problem



# Project 2 - Safe Museum Tour Problem

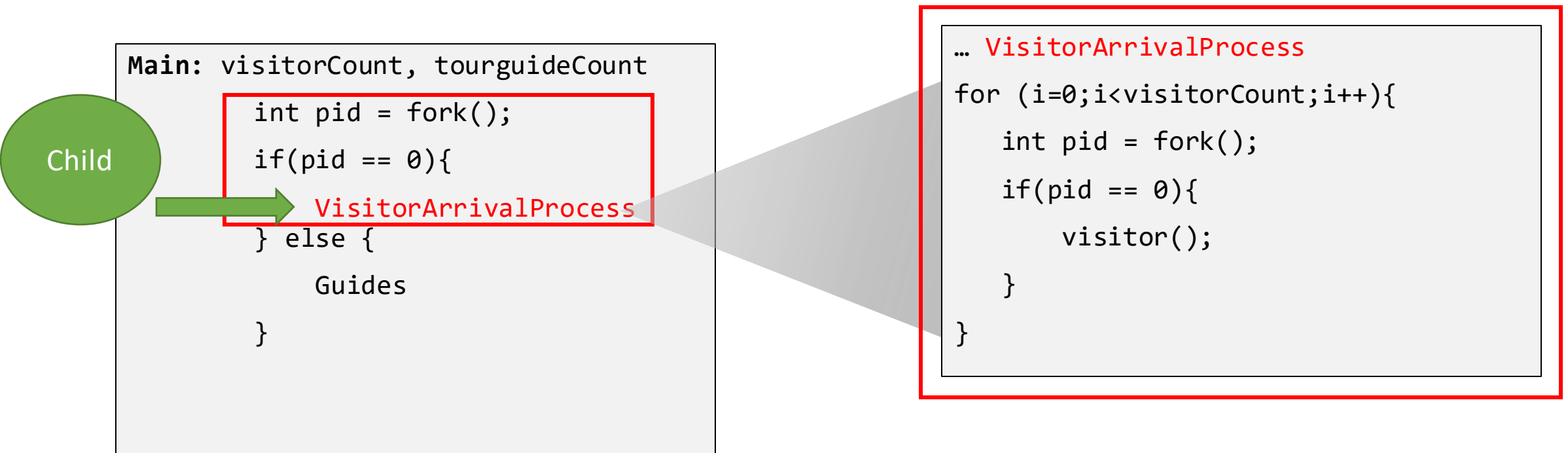


# Project 2 - Safe Museum Tour Problem



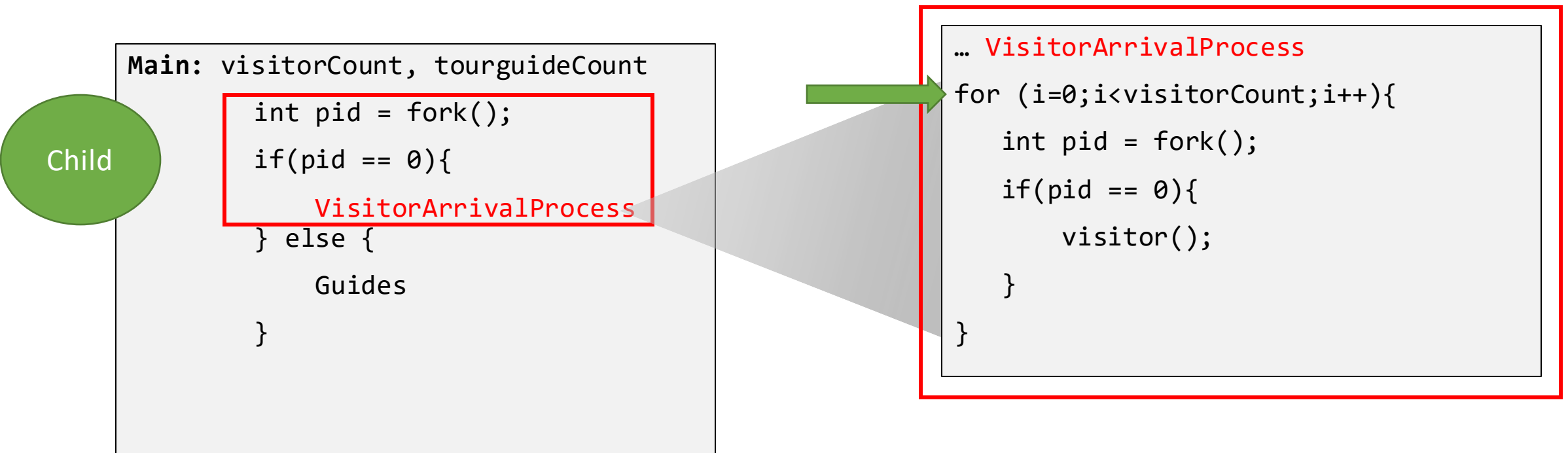
# Project 2 - Safe Museum Tour Problem

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival



# Project 2 - Safe Museum Tour Problem

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival



# Project 2 - Safe Museum Tour Problem

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival

Child

```
Main: visitorCount, tourguideCount
    int pid = fork();
    if(pid == 0){
        VisitorArrivalProcess
    } else {
        Guides
    }
```

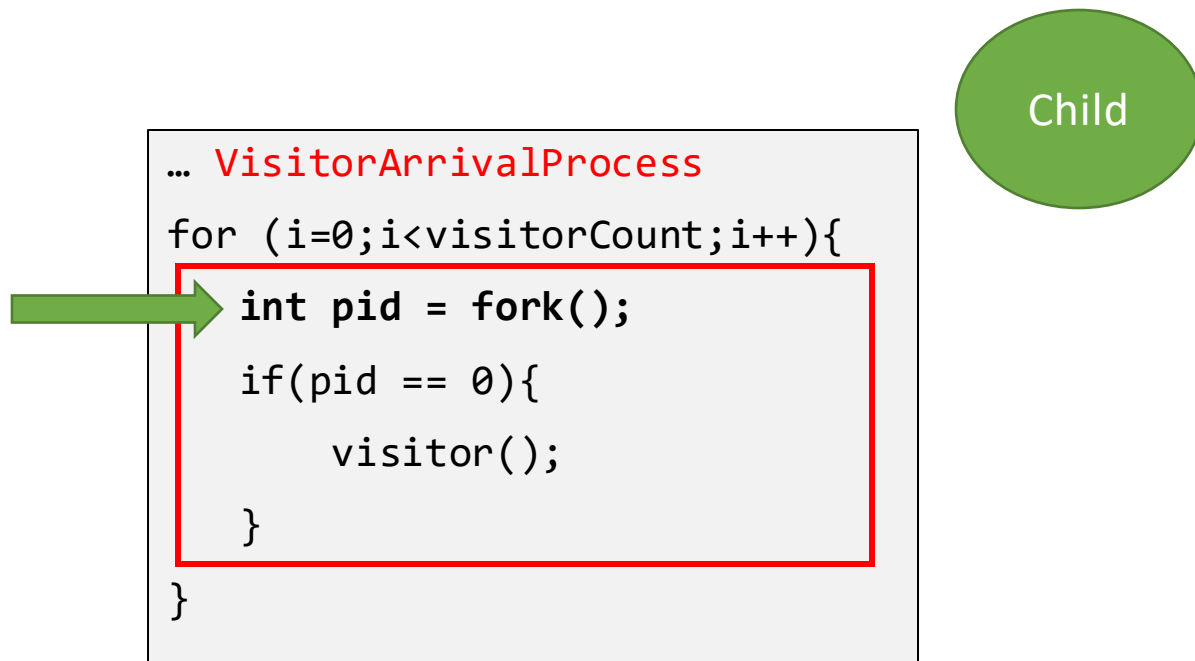
```
... VisitorArrivalProcess
for (i=0;i<visitorCount;i++){
    int pid = fork();
    if(pid == 0){
        visitor();
    }
}
```



# Project 2 - Safe Museum Tour Problem

---

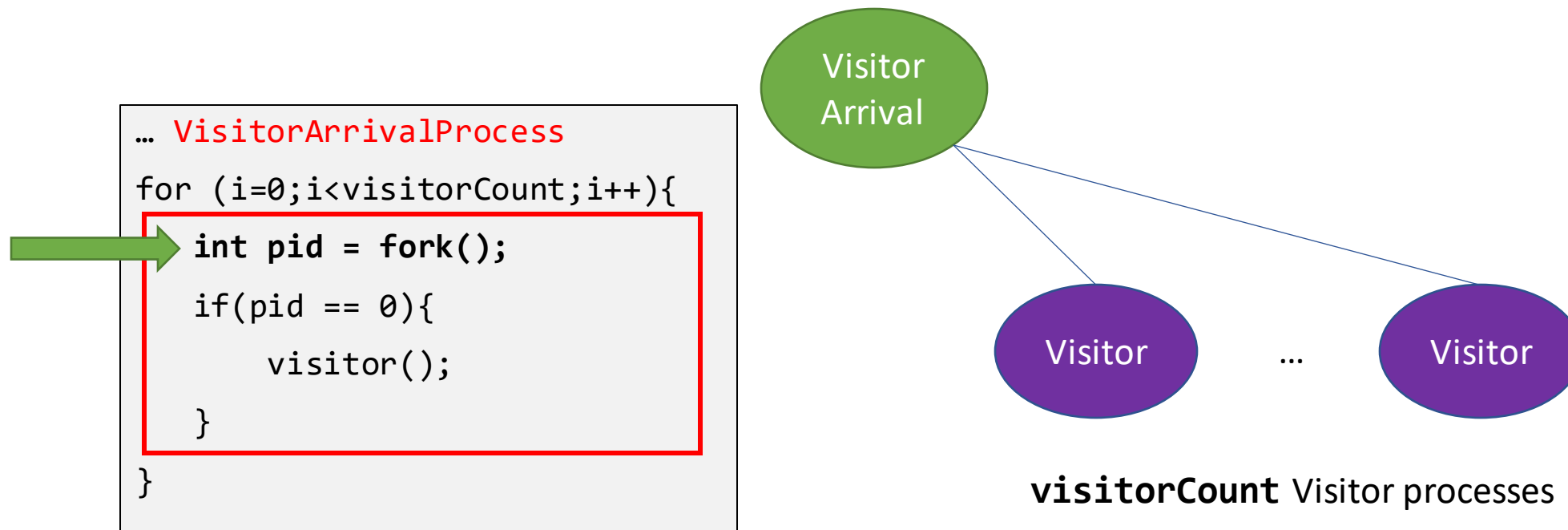
- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival



# Project 2 - Safe Museum Tour Problem

---

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival



# Project 2 - Safe Museum Tour Problem

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival

Visitor  
Arrival

```
... VisitorArrivalProcess
for (i=0;i<visitorCount;i++){
    int pid = fork();
    if(pid == 0){
        visitor();
    }
}
```

Visitor

```
... VisitorArrivalProcess
for (i=0;i<visitorCount;i++){
    int pid = fork();
    if(pid == 0){
        visitor();
    }
}
```

# Project 2 - Safe Museum Tour Problem

---

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival

Visitor  
Arrival

```
... VisitorArrivalProcess
for (i=0;i<visitorCount;i++){
    int pid = fork();
    if(pid == 0){
        visitor();
    }
}
```

Visitor

```
... VisitorArrivalProcess
for (i=0;i<visitorCount;i++){
    int pid = fork();
    if(pid == 0){
        visitor();
    }
}
```

# Project 2 - Safe Museum Tour Problem

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival

Visitor  
Arrival

```
... VisitorArrivalProcess
for (i=0;i<visitorCount;i++){
    int pid = fork();
    if(pid == 0){
        visitor();
    }
}
```

Visitor

```
... VisitorArrivalProcess
for (i=0;i<visitorCount;i++){
    int pid = fork();
    if(pid == 0){
        visitor();
    }
}
```

# Project 2 - Safe Museum Tour Problem

---

Visitor  
Arrival

```
... VisitorArrivalProcess
for (i=0;i<visitorCount;i++){
    int pid = fork();
    if(pid == 0){
        visitor();
    }
}
```

# Project 2 - Safe Museum Tour Problem

Visitor  
Arrival

... **VisitorArrivalProcess**

```
srand(seed);
for (i=0;i<visitorCount;i++){
    int pid = fork();
    if(pid == 0){
        visitor();
    }else {
        int value = rand() % 100 + 1; //random number between 1-100
        if (value > 50){sleep(delay)};
    }
}
```

The visitor has to sleep for  
some **time** with 50%  
probability

# Project 2 - Safe Museum Tour Problem

Visitor  
Arrival

... **VisitorArrivalProcess**

```
srand(seed);
for (i=0;i<visitorCount;i++){
    int pid = fork();
    if(pid == 0){
        visitor();
    }else {
        int value = rand() % 100 + 1; //random number between 1-100
        if (value > 50){sleep(delay)};
    }
    for (i=0;i<visitorCount;i++){
        wait(NULL);
    }
}
```

Then after creating all the visitors it will **wait** for them to finish.



# Project 2 - Safe Museum Tour Problem

---

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival

# Project 2 - Safe Museum Tour Problem

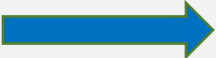
---

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to **simulate guides** and visitor's arrival

# Project 2 - Safe Museum Tour Problem

- Two processes to **simulate guides** and visitor's arrival

Main

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
     for (i=0; i<guideCount; i++){
        int pid = fork();
        if(pid == 0){
            guides();
        }
    }
}
```

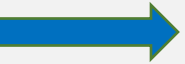
The parent will **create child processes** for the guide **directly**

# Project 2 - Safe Museum Tour Problem

---

- Two processes to **simulate guides** and visitor's arrival

Main

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    for (i=0; i<guideCount; i++){
         int pid = fork();
        if(pid == 0){
            guides();
        }
    }
}
```

# Project 2 - Safe Museum Tour Problem

Main

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    srand(seed);
    for (i=0; i<guideCount; i++){
        int pid = fork();
        if(pid == 0){
            guides();
        }
        int value = rand() % 100 + 1;
        if (value > 50){sleep(delay)};
    }
}
```

However, he also **must**  
sleep with **50% chance**

# Project 2 - Safe Museum Tour Problem

Main

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    srand(seed);
    for (i=0; i<guideCount; i++){
        int pid = fork();
        if(pid == 0){
            guides();
        }
        int value = rand() % 100 + 1;
        if (value > 50){sleep(delay)};
    }
}
```

Child  
Guide

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    srand(seed);
    for (i=0; i<guideCount; i++){
        int pid = fork();
        if(pid == 0){
            guides();
            exit()
        }
        int value = rand() % 100 + 1;
        if (value > 50){sleep(delay)};
    }
}
```

# Project 2 - Safe Museum Tour Problem

Main

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    srand(seed);
    for (i=0; i<guideCount; i++){
        int pid = fork();
        if(pid == 0){
            guides();
        }
        int value = rand() % 100 + 1;
        if (value > 50){sleep(delay)};
    }
}
```

Child  
Guide

```
int pid = fork();
if(pid == 0){
    VisitorArrivalProcess
} else {
    srand(seed);
    for (i=0; i<guideCount; i++){
        int pid = fork();
        if(pid == 0){
            guides();
            exit();
        }
        int value = rand() % 100 + 1;
        if (value > 50){sleep(delay)};
    }
}
```

# Project 2 - Safe Museum Tour Problem

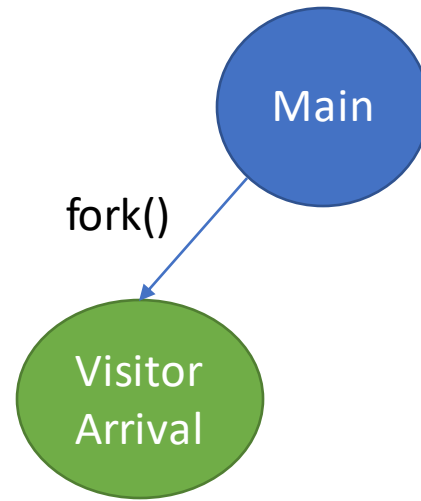
---

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival



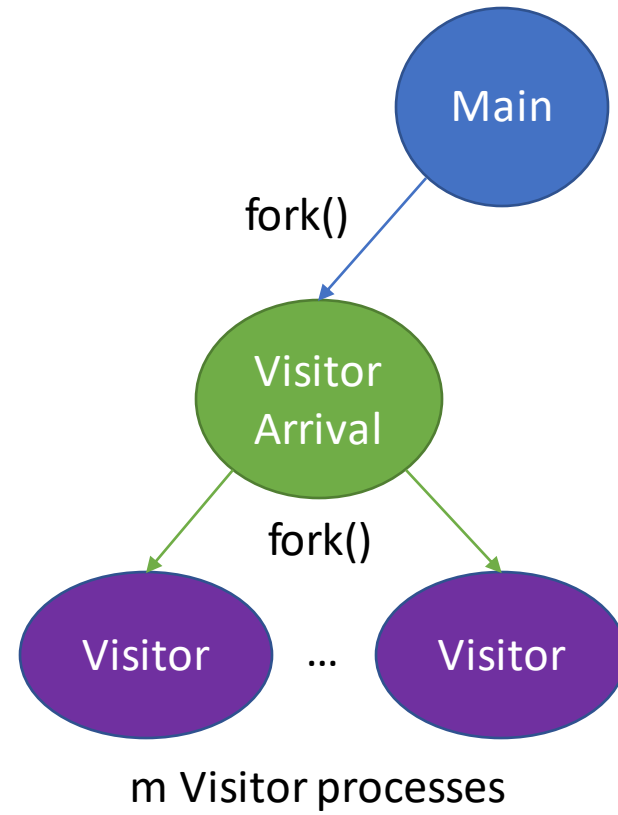


Each node **is**  
**a process**

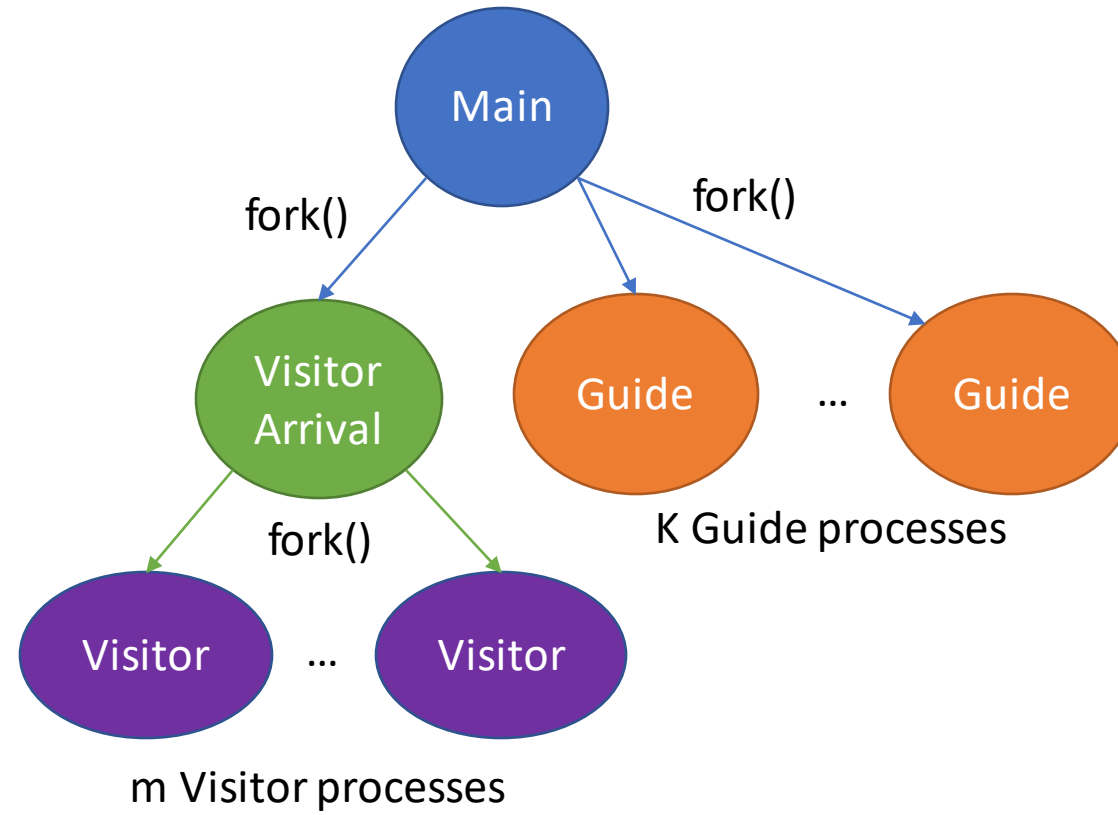


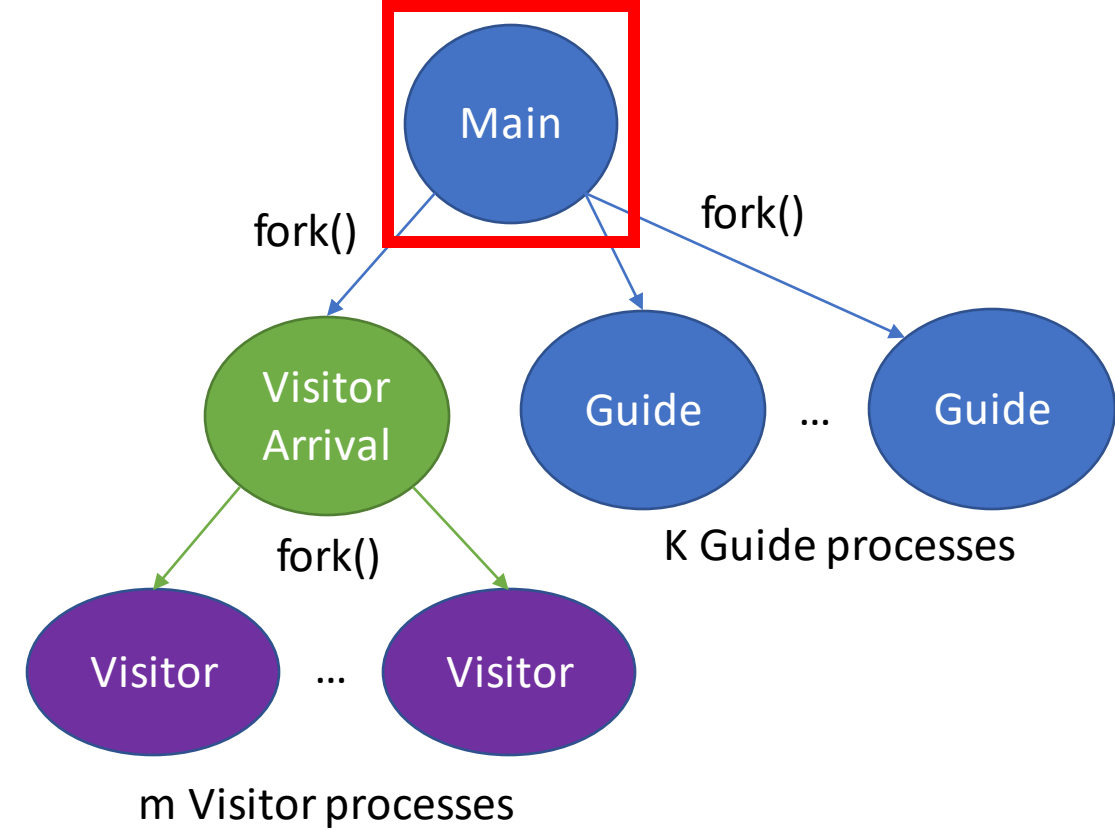
Each node **is**  
**a process**

Each node **is**  
**a process**



Each node **is**  
**a process**

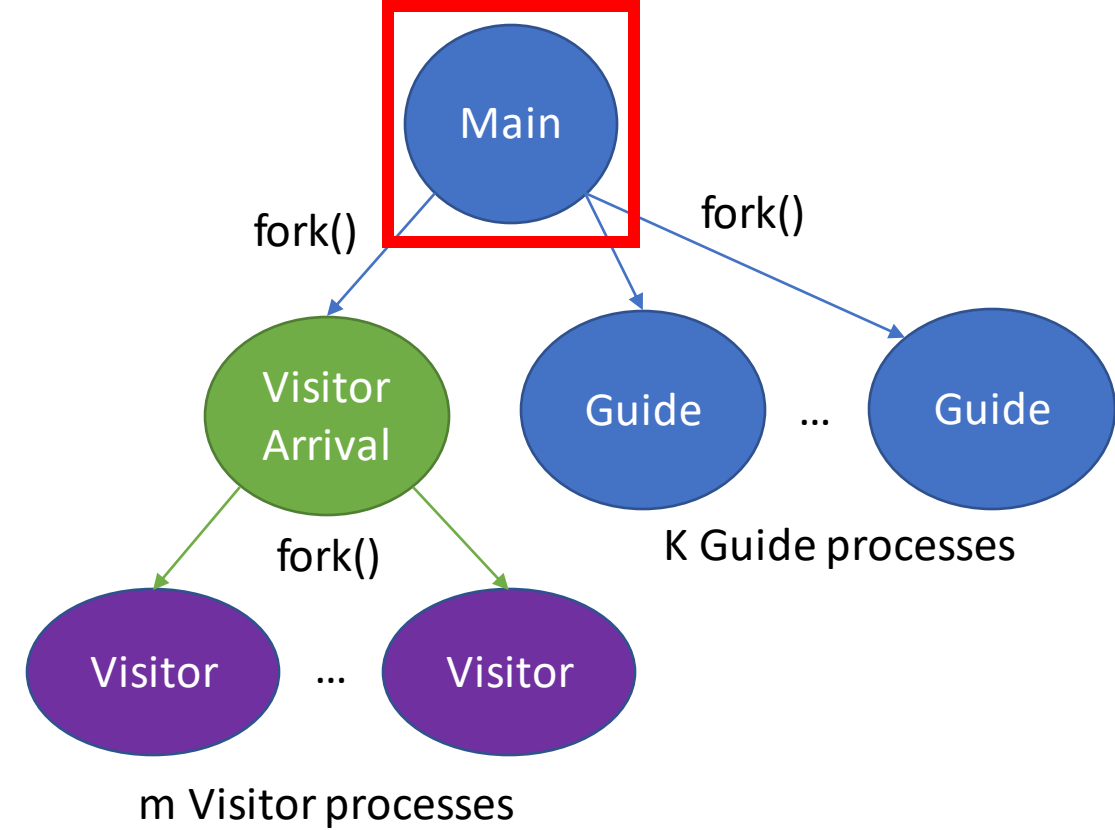




Main:

```
int t0 = getTimeOfDay(); //1. get initial time stamp
int pid = fork();        // 2. create child process (visitor arrival)
if(pid == 0){             //child process when pid == 0
    VisitorArrivalProcess //visitor arrival process
} else {                  //original process when pid != 0
    GuideCriations         //3. create following child process(guides)
}
```





Main:

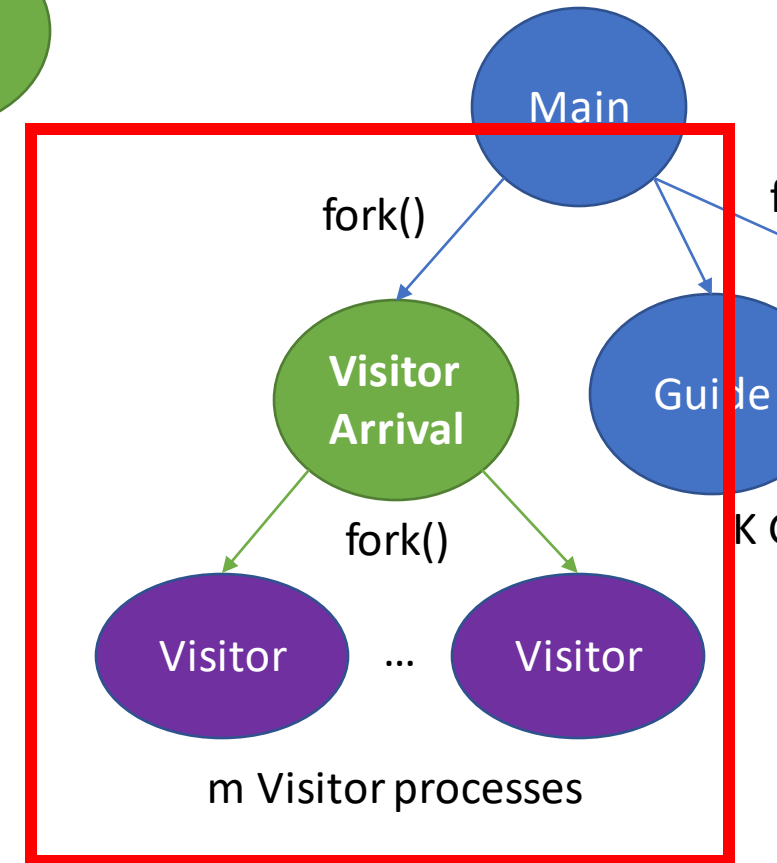
```
int t0 = getTimeOfDay(); //1. get initial time stamp
int pid = fork();        // 2. create child process (visitor arrival)
if(pid == 0){             //child process when pid == 0
    VisitorArrivalProcess //visitor arrival process
} else {                  //original process when pid != 0
    GuideCriations        //3. create following child process(guides)
}
```

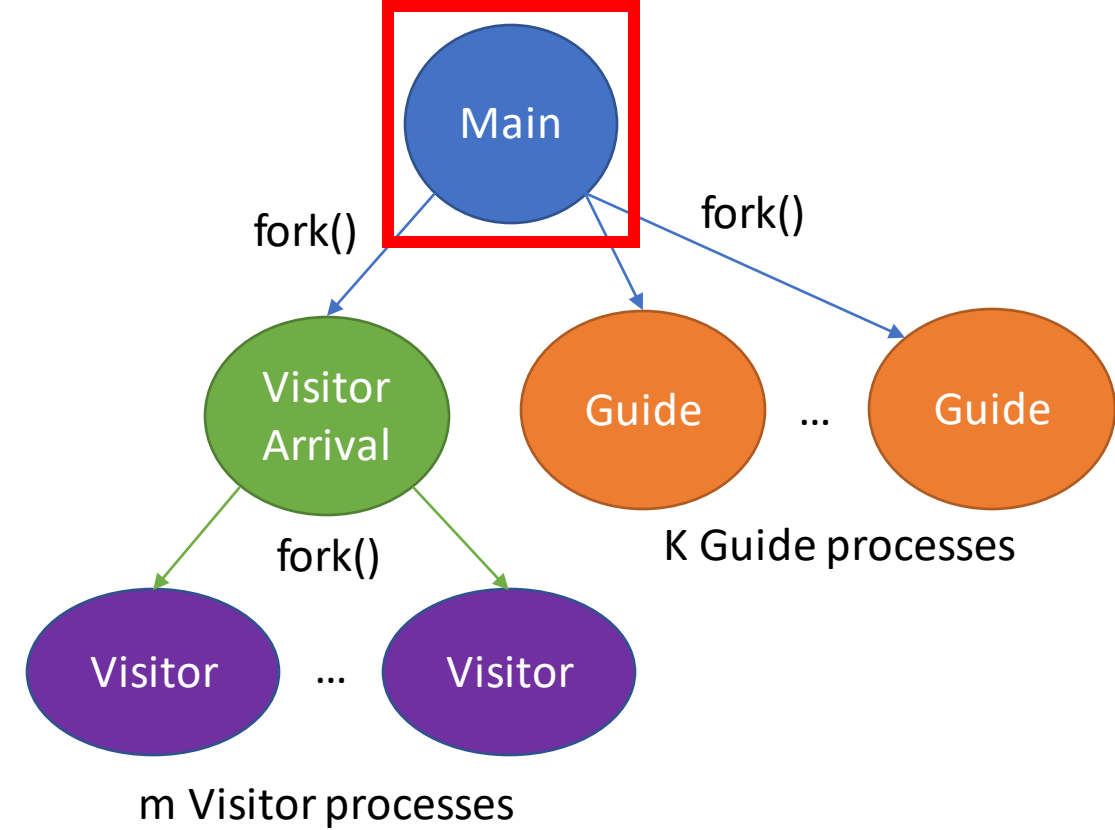


VisitorArrivalProcess :

```
srand(seed); //1. setup random seed for random arrival
for (int i = 0; i < m; i++){ //2. fork child processes by flip a coin
    int pid2 = fork();           //fork child processes (visitors)
    if(pid2==0){                 //child process (visitors) when pid2==0
        visitors();              // Visitor
        exit();
    } else {                    //original process(VisitorArrivalProcess) when pid2!=0
        int value = rand() % 100 + 1; //random number between 1-100
        if (value > 50){sleep(delay)};
    }
}
for (int i = 0; i < m; i++){ //3. wait until all children finished
    wait(NULL);
}
```

Visitor  
Arrival



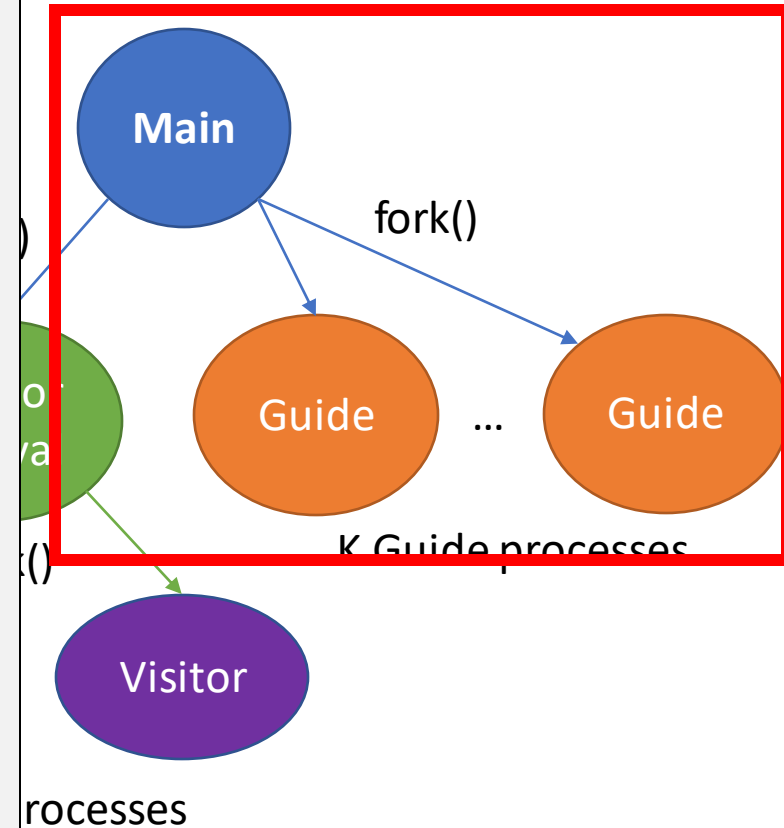


```
Main:
int t0 = getTimeOfDay(); //1. get initial time stamp
int pid = fork();        // 2. create child process (visitor arrival)
if(pid == 0){             //child process when pid == 0
    VisitorArrivalProcess //visitor arrival process
} else {                  //original process when pid != 0
    GuideCriations         //3. create following child process(guides)
}
```



GuideCriaions :

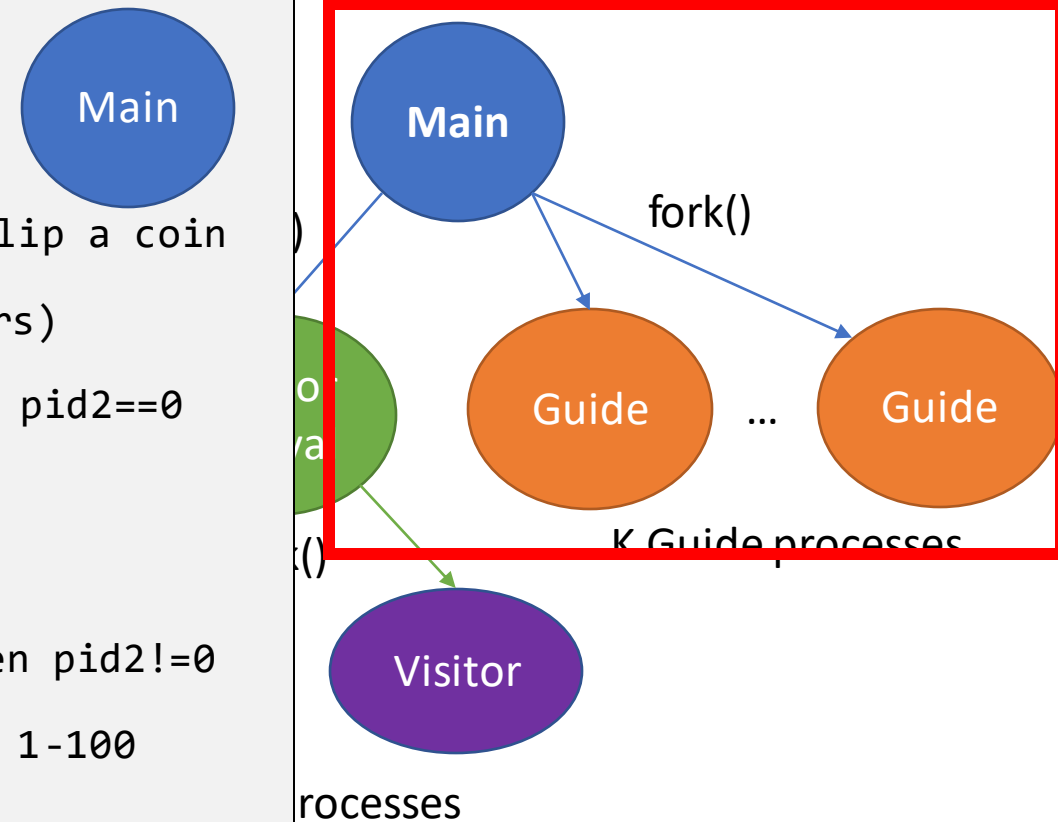
```
srand(seed); //1. setup random seed for random arrival
for (int i = 0; i < m; i++){ //2. fork child processes by flip a coin
    int pid2 = fork();           //fork child processes (visitors)
    if(pid2==0){                 //child process (visitors) when pid2==0
        guide();                 //child process (visitors) when pid2==0
        exit();
    } else {                    //original process(VisitorArrivalProcess) when pid2!=0
        int value = rand() % 100 + 1; //random number between 1-100
        if (value > 50){sleep(delay)};
    }
}
for (int i = 0; i < m; i++){ //3. wait until all children finished
    wait(NULL);
}
wait(NULL)
```

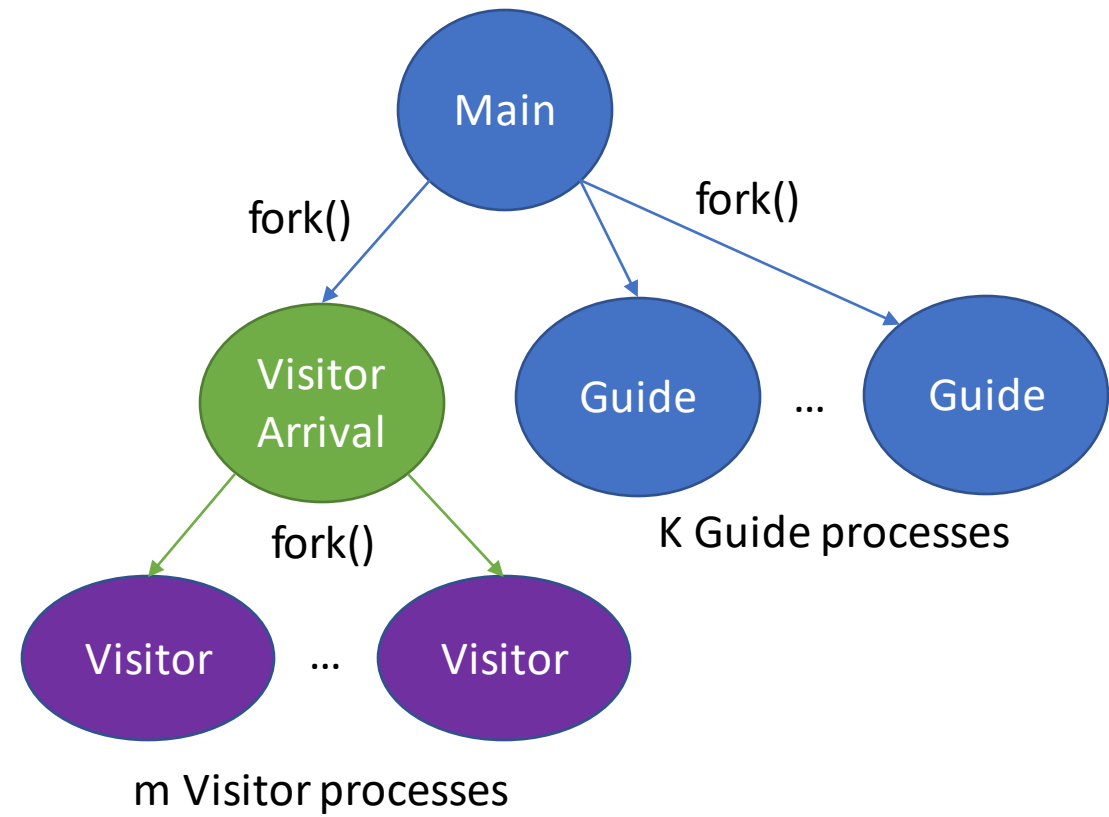


GuideCriaions :

```
srand(seed); //1. setup random seed for random arrival
for (int i = 0; i < m; i++){ //2. fork child processes by flip a coin
    int pid2 = fork();           //fork child processes (visitors)
    if(pid2==0){                 //child process (visitors) when pid2==0
        guide();                 //child process (visitors) when pid2==0
        exit();
    } else {                    //original process(VisitorArrivalProcess) when pid2!=0
        int value = rand() % 100 + 1; //random number between 1-100
        if (value > 50){sleep(delay)};
    }
}
for (int i = 0; i < m; i++){ //3. wait until all children finished
    wait(NULL);
}
```

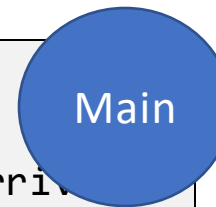
**wait(NULL) Parent has to wait for the VisitorArrivalProcess**





```

Main:
int t0 = getTimeOfDay(); //1. get initial time stamp
int pid = fork();        // 2. create child process (visitor arrival)
if(pid == 0){             //child process when pid == 0
    VisitorArrivalProcess //visitor arrival process
} else {                  //original process when pid != 0
    GuideCriations        //3. create following child process(guides)
}
wait(Null)
  
```



# Project 2 - Safe Museum Tour Problem

---

- The visitor **process** (Program Parameter)
- The tour guide **process** (Program Parameter)
- Two processes to simulate guides and visitor's arrival
- Six functions called by these processes
  - visitorArrives()
  - tourMuseum()
  - visitorLeaves()
  - tourguideArrives()
  - openMuseum()
  - tourGuideLeaves()

# Project 2 - Safe Museum Tour Problem

---

- For the museum to open
  - 1 tour guide
  - 1 Visitor
  - visitorArrives() is called by **visitor process**
  - tourGuideArrives() is called by **tour guide**

# Project 2 - Safe Museum Tour Problem

---

- A **visitor** must wait if
  - Museum is closed
  - Max visitors is reached

# Project 2 - Safe Museum Tour Problem

---

- A visitor must wait if
  - Museum is closed
  - Max visitors is reached
- An arriving tour guide **must wait if**
  - Museum is closed and no visitor is waiting, **or**
  - Museum is open, and **two tour guides** are inside the museum

# Project 2 - Safe Museum Tour Problem

---

- A visitor must wait if
  - Museum is closed
  - Max visitors is reached
- An arriving tour guide must wait if
  - Museum is closed and no visitor is waiting, **or**
  - Museum is open, and two tour guides are inside the museum
- While only **one tour guide** is inside the museum
  - Up to ten visitors may arrive (i.e., calling visitorArrives())
  - Second tour guide may open the museum (i.e., call openMuseum())



# Project 2 - Safe Museum Tour Problem

---

- When a tour arrives you should print
  - **Tour guide %d arrives at time d%**
- When a visitor arrives you print
  - **Visitor %d arrives at time %d**
- Remember to use these calls to print:

```
fprintf(stderr,<print content here>)
```

OR

```
printf(<print content here>);  
fflush(stdout);
```

# Project 2 - Safe Museum Tour Problem

---

- Memory allocation

**#include <sys/mman.h>**

**void \*mmap(void \**addr*, size\_t *length*, int *prot*, int *flags*, int *fd*, off\_t *offset*);**

# Project 2 - Safe Museum Tour Problem

---

- Memory allocation

```
#include <sys/mman.h>
```

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

- set flags=**MAP\_SHARED** when allocating shared memory

# Project 2 - Safe Museum Tour Problem

---

- Memory allocation

```
#include <sys/mman.h>
```

```
void *mmap(void *addr, size_t length, int prot, int flags, int fd, off_t offset);
```

- set flags=**MAP\_SHARED** when allocating shared memory

- Usage e.g.:

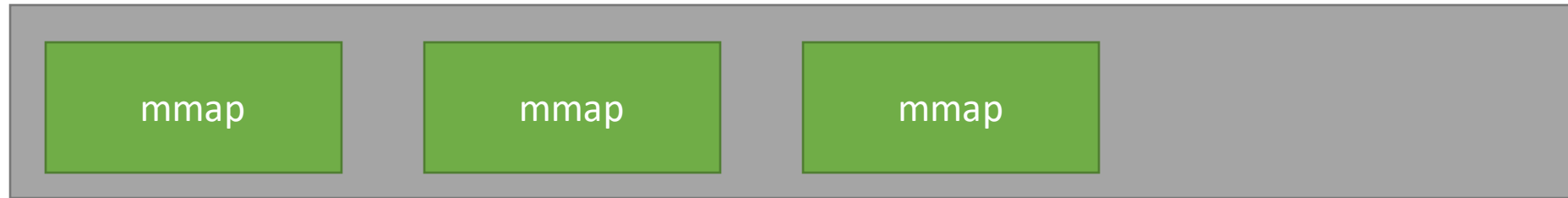
```
int* const = (int*) mmap(NULL, sizeof(int), PROT_READ|PROT_WRITE,  
MAP_SHARED|MAP_ANONYMOUS, 0, 0);
```

- More info: <http://man7.org/linux/man-pages/man2/mmap.2.html>

# Project 2 - Safe Museum Tour Problem

---

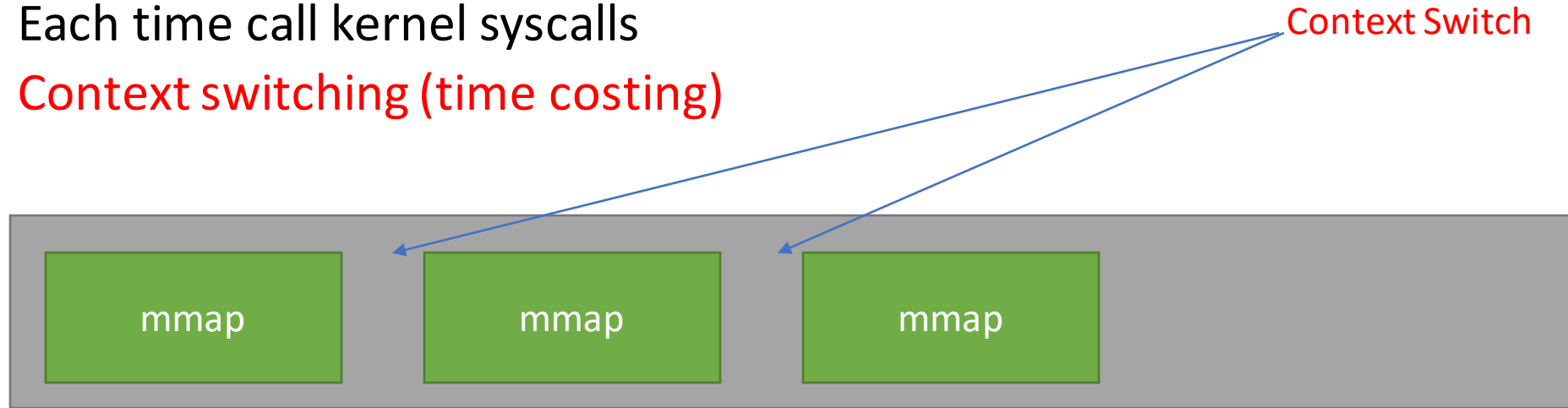
- Memory allocation
  - Each time call kernel syscalls



# Project 2 - Safe Museum Tour Problem

---

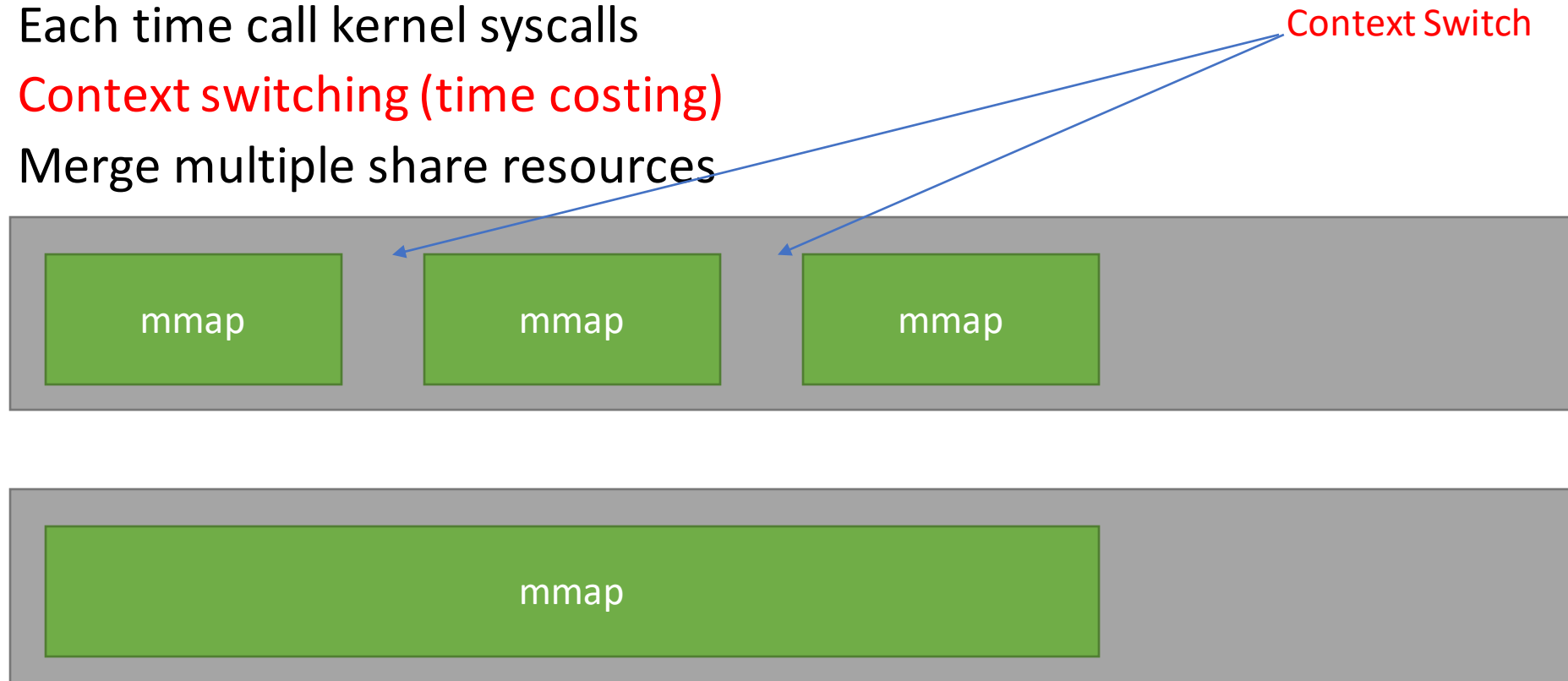
- Memory allocation
  - Each time call kernel syscalls
  - Context switching (time costing)



# Project 2 - Safe Museum Tour Problem

---

- Memory allocation
  - Each time call kernel syscalls
  - Context switching (time costing)
  - Merge multiple share resources



# Project 2 - Safe Museum Tour Problem

---

- Memory allocation
  - Each time call kernel syscalls
  - Context switching (time costing)
  - Merge multiple share resources

```
struct shared_mem{  
    int visitors;  
    int guides;  
    struct cs1550_sem sem1;  
    ...  
}
```





# CS 1550

Week 5 - Project 2 Discussion

Teaching Assistant

Henrique Potter