Exercise 00 ~(20min)

Really good read about threads. **<- You can search this if you like.**

Lets Talk Threads

Why use *threads*? ~ (5min)

Answer

> 1. Creating a thread goes 10–100 times faster than creating a process.
> 2. To shear address space
> 3. Performance
> 4. To work with blocking system calls
> 5. There is a blocking call

What are threads? ~ (5min)

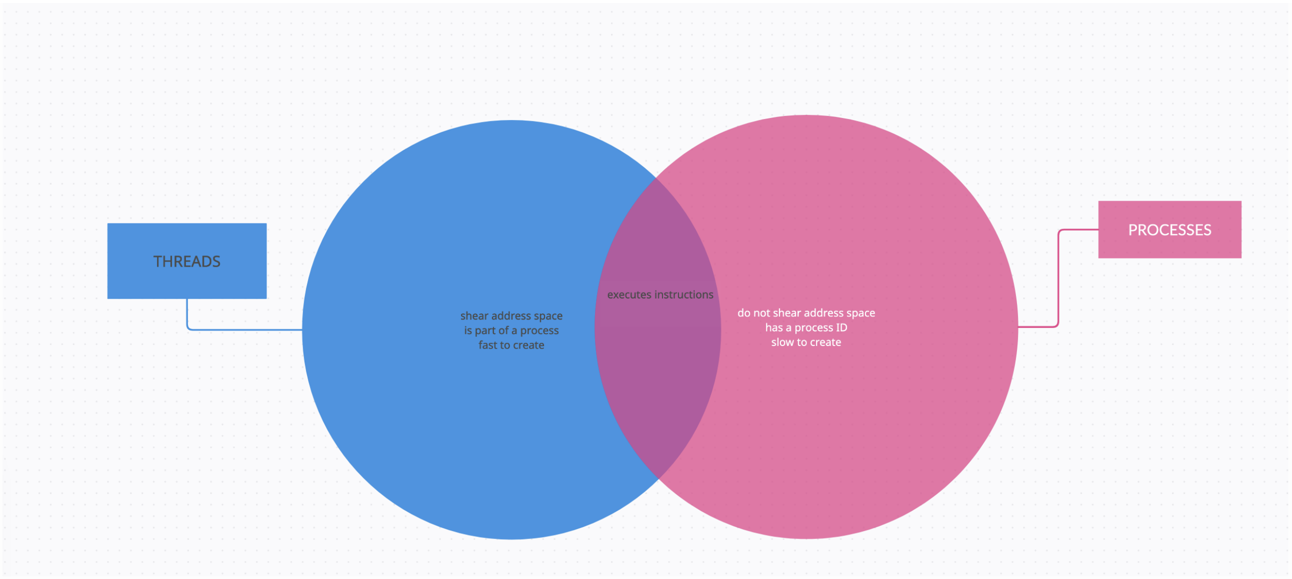
Answer

> One way of looking at a process is that it is a way to group related resources. A process has an address space containing program text and data and other resources. These resources may include open files, child processes, pending alarms, signal handlers, accounting information, and more. By putting them together in the form of a process, they can be managed more easily. The other concept a process has is a **thread** of execution, usually shortened to just **thread**. The **thread** has a program counter that keeps track of which instruc-tion to execute next. It has registers, which hold its current working variables. It has a stack, which contains the execution history, with one frame for each proce-dure called but not yet returned from. Although a **thread** must execute in some process, the **thread** and its process are different concepts and can be treated sepa-rately. Processes are used to group resources together; **threads** are the entities scheduled for execution on the CPU.

What are the differences between processes and threads ~ 5(min)

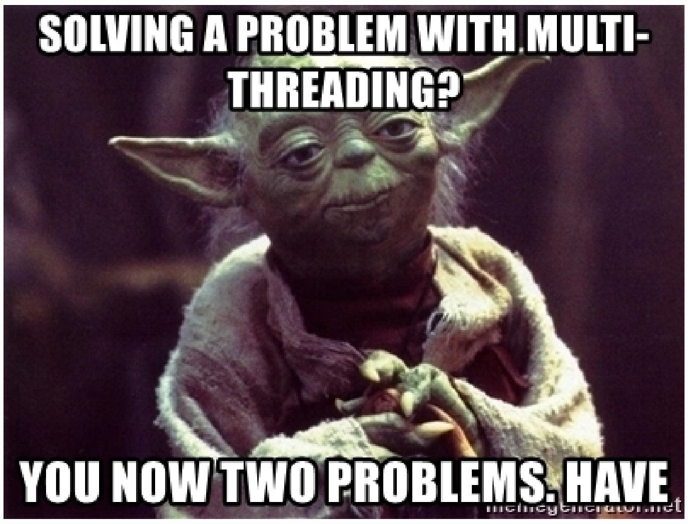
Does not have to be exactly like this but make sure that every one gets the difference bewteen threads and processes

Answer



Ex 01 ~(20min)

Preamble:


SOLVING A PROBLEM WITH MULTI-THREADING?
YOU NOW TWO PROBLEMS. HAVE
memegenerator.net

Description:

Create a thread, see what it does, how it works.

Goal to achieve:

Create a new thread that outputs this message!

```
$ ./ex01.out
Hi From thread. You can call me philosopher 0
```

Allowed functions:

> printf
>
> pthread_create,
>
> ## pthread_join
>
> ### what is pthread_t ~(5min)
>
> 1. what data type is this?
>
> ### pthread_create ~ (5min)
>
> 1. How does the prototype look like?
> 2. What arguments does the function take?
> 3. What is void *(*start_routine)(void *)?
> 4. How would you pass data to the start_routine function?
> 5. What is the attr argument?
>
> ### pthread_join ~ (5min)
>
> 1. How does the prototype look like?
> 2. What arguments does the function take?
>
> ## 3. What is void **value_ptr used for?
>
> **Answer**

```c
#include <stdio.h>
#include <pthread.h>

void    *routine(void *ptr)
{
    printf("Hi From thread. You can call me philosopher 0\n");
    return (NULL);
}

int    main()
{
    pthread_t    thread;

    pthread_create(&thread, NULL, routine, NULL);
    pthread_join(thread, NULL);
    return (0);
}
```

When your programme is ready run `unit_test.sh`.

Ex02 ~(5min - 10min)

Goal:

creat 20 threads that will print the following

```
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
Hi From thread. You can call me philosopher 0
```

Answer

```c

#include <stdio.h>
#include <pthread.h>

#define MAX_PHILO 20

void    *rutine(void *ptr)
{
    printf("Hi From thread. You can call me philosopher 0\n");
    return (NULL);
}

int    main()
{
    pthread_t    thread[MAX_PHILO];

    for (int i = 0; i < MAX_PHILO; i++)
    {
        pthread_create(&thread[i], NULL, rutine, NULL);
    }
    for (int i = 0; i < MAX_PHILO; i++)
    {
        pthread_join(thread[i], NULL);
    }
    return (0);
}
```

Ex03 ~35min

Race conditions

Whatch a video about data races

1. What are race conditions?
2. What is a critical section?

How to spot race conditions?

Answer

`-fsanitize=thread`

Do you see a data race in this code?

```c
// example code
#include <stdio.h>
#include <pthread.h>

void    *rutine(void *ptr)
{
    while (*(int *)ptr < 1000)
    {
        *(int *)ptr += 1;
    }
    printf("Done\n");
    return (NULL);
}

int    main()
{
    pthread_t    thread;
    int          index;

    index = 0;
    pthread_create(&thread, NULL, rutine, &index);
    while (index < 10000)
    {
        index++;
    }
    pthread_join(thread, NULL);
    return (0);
}
```

What are mutexes?

What is the pthread_mutex_t data type?

pthread_mutex_init ~ 5(min)

What does this function do?

pthread_mutex_destroy ~ 5(min)

```
What does this function do?
Do you have to free the mutex?
```

pthread_mutex_lock ~ 5 (min)

```
What does this function do?
Can you lock a mutex that is not inited?
```

pthread_mutex_unlock ~ 25 (min)

```
What does this function do?
What happens when u unlock a mutex 2times?
```

Goal:
Make a program that inits, locks, unlock and destroys a mutex!

```
#include <pthread.h>

int    main()
{
    pthread_mutex_t lock;

    pthread_mutex_init(&lock, NULL);
    pthread_mutex_lock(&lock);
    pthread_mutex_unlock(&lock);
    pthread_mutex_destroy(&lock);
    return (0);
}
```

Goal:
Prevent the data race in the example code.

ex04

Deadlocks

What is a deadlock?
When does a deadlock occur?

Goal:

Goal:
Produce a program that has a deadlock.

Answer

```
#include <pthread.h>

int     main()
{
    pthread_mutex_t lock;

    pthread_mutex_init(&lock, NULL);
    pthread_mutex_lock(&lock);
    pthread_mutex_lock(&lock);
    pthread_mutex_unlock(&lock);
    pthread_mutex_destroy(&lock);
    return (0);
}
```

Break

ex05

creat 20 threads that will print the following
the order does not matter

```
Hi From thread. You can call me philosopher 1
Hi From thread. You can call me philosopher 2
Hi From thread. You can call me philosopher 3
Hi From thread. You can call me philosopher 4
Hi From thread. You can call me philosopher 5
Hi From thread. You can call me philosopher 6
Hi From thread. You can call me philosopher 7
Hi From thread. You can call me philosopher 8
Hi From thread. You can call me philosopher 9
Hi From thread. You can call me philosopher 10
Hi From thread. You can call me philosopher 11
Hi From thread. You can call me philosopher 12
Hi From thread. You can call me philosopher 13
Hi From thread. You can call me philosopher 14
Hi From thread. You can call me philosopher 15
Hi From thread. You can call me philosopher 16
Hi From thread. You can call me philosopher 17
Hi From thread. You can call me philosopher 18
Hi From thread. You can call me philosopher 19
Hi From thread. You can call me philosopher 20
Hi From thread. You can call me philosopher 21
```

Answer

```c
#include <stdio.h>
#include <pthread.h>

typedef struct s_list
{
    int     index;
    pthread_mutex_t lock;
}    t_list;

#define MAX_PHILO 20

void    *rutine(void *ptr)
{
    pthread_mutex_lock(&((t_list *)ptr)->lock);
    printf("Hi From thread. You can call me philosopher %d\n", (*(t_list *)ptr).index +
    pthread_mutex_unlock(&((t_list *)ptr)->lock);
    return (NULL);
}

int    main()
{
    pthread_t   thread[MAX_PHILO];
    t_list      philosopher[MAX_PHILO];

    for (int i = 0; i < 20; i++)
    {
        pthread_mutex_init(&philosopher[i].lock, NULL);
        pthread_mutex_lock(&philosopher[i].lock);
        philosopher[i].index = i;
        pthread_mutex_unlock(&philosopher[i].lock);
        pthread_create(&thread[i], NULL, rutine, &philosopher[i]);
    }
    for (int i = 0; i < MAX_PHILO; i++)
    {
        pthread_join(thread[i], NULL);
        pthread_mutex_destroy(&philosopher[i].lock);
    }
    return (0);
}
```

Bonus

Make a program that will use 3 created threads to add up an int to 42. Threads can increment the int every .5sec
once the value is 42 the program has to print "Got it!\n" and exit.

catch:
The threads do not know when the value is 42

What is a monitoring thread?

A monitoring thread is a concept used in the philosopher's project. a monitoring thread will keep track of the int variable. If the value is 42 let the threads know to finish and exit.

```
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>
#include <stdbool.h>

typedef struct    s_data
{
    bool            is_done;
    pthread_mutex_t mutex_lock;
}                 t_data;

void    *rutine(void *ptr)
{
    t_data  *data = ptr;

    while (1)
    {
        pthread_mutex_lock(&data->mutex_lock);
        if (data->is_done)
        {
            printf("I am dead\n");
            pthread_mutex_unlock(&data->mutex_lock);
            return (NULL);
        }
        pthread_mutex_unlock(&data->mutex_lock);
        sleep(1);
        printf("I am still alive\n");
    }
    return (NULL);
}
```

```
int    main()
{
    pthread_t       thread[4];
    t_data          data;


    data.is_done = false;
    pthread_mutex_init(&data.mutex_lock, NULL);
    for (int i = 0; i < 4; i++)
        pthread_create(&thread[i], NULL, rutine, &data);
    sleep(5);
    pthread_mutex_lock(&data.mutex_lock);
    data.is_done = true;
    pthread_mutex_unlock(&data.mutex_lock);
    for (int i = 0; i < 4; i++)
        pthread_join(thread[i], NULL);
    pthread_mutex_destroy(&data.mutex_lock);
    return (0);
}
```