CSE-302 15 Jan 2024/Mo

B: Unix System Programming ....

System Programming Notes:

chapter 12: Unix/POSIX Threads.

→ One method of achieving parallelism is for multiple process to cooperate and synchronize throug shared memory and message passing

→ An alternative approach uses multiple threads of execution in a single address space.

12.1 → A motivating Problem: Monitoring File descriptors:

→ Six general approaches to monitor multiple file descriptors for input under POSIX are as follows.

1 A seperat process (fork) monitors each file descriptor.

Problem: since the child does't share any variable, we may use shared memory or message passing to exchange information

2 Using select (2) system call.
3 Using poll system call. } blocking calls.

Problem: Once the blocking call returns, the calling program handles each ready file descriptor in turn. Furthmore, the program can do no useful work.

4 Non-blocking I/O with polling. work well when the program has to do "usefull' work" that it can perform between its intermittents checks to see if I/O is available.

problem: most problems are difficult to structure in this way, and it somtimes forces hard-coding of the timing for I/O check relative to useful work. otherwis it can lead to busy waiting.

5 POSIX asynchronous I/O can be used with or without signal's notification.

Problems: If used with signals

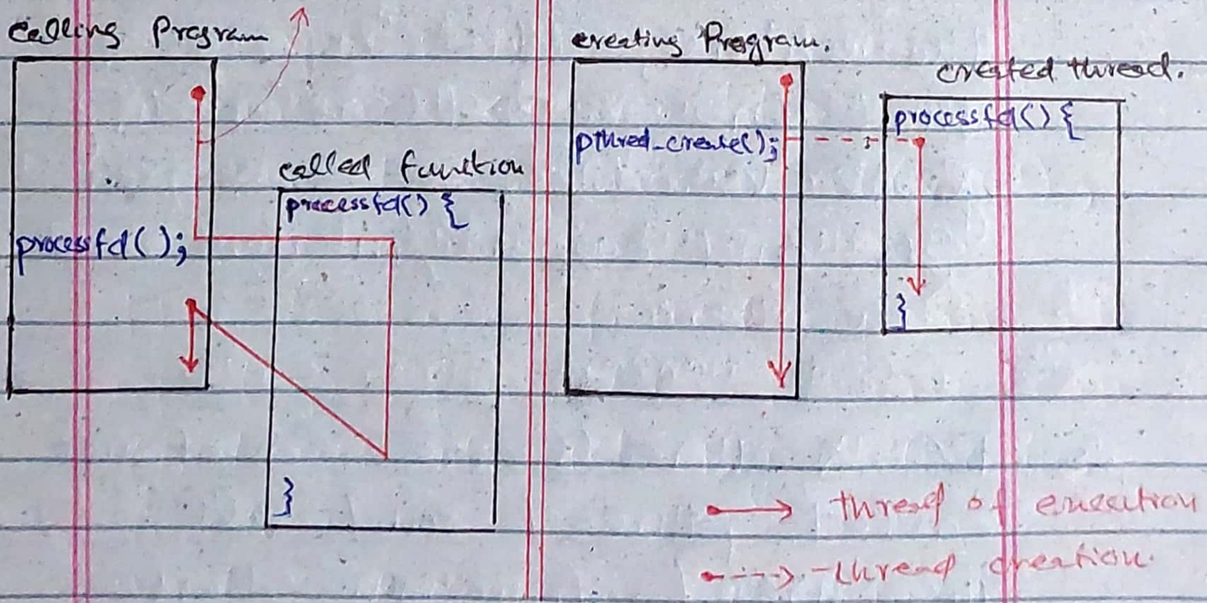
the handler used should by async-signal-safe functions. It (handler) can cause potential for deadlocks and race conditions when synchronizing with other/rest of program. the approach is error prone and difficult to implement.

6 A seperate thread monitors each FD. This approach is simpler and program can overlap processing with waiting for input in a transparent way.

12.2 Use of Threads to monitor multiple file descriptors:

→ Single thread of Execution.

→ Two threads of execution.



→ The thread is "schedulable entity" with its own value of the program counter, its own stack and its own scheduling parameters. It executes an independent stream of instructions, never returning to the point of call.

12.3 Thread Management:

→ A thread package usually includes functions for thread creation, destruction, scheduling, enforcement of mutual exclusion and conditional waiting, run-time system to manage threads transparently.

→ the thread for a process share the entire address space of that process. and can modify global variables, access open fds and cooperat/interfere with each other.

→ POSIX threads are sometimes called pthread and all the thread functions start with 'pthreads'

→ Some posix threads mangment functions are:

#	POSIX function	Description
1	pthread_cancel	terminate another thread.
2	" _create	create a thread.
3	" _detach	set thread to release resources.
4	" _equal	test two threads IDs for equility.
5	" _exit	exit a-thread without exiting proc.
6	" _kill	send a signal to a thread.
7	" _join	wait for a thread.
8	" _self	find out own thread ID.

→ Most pthreads returns 0 if successful and nonzero error code if unsuccessful. (don't set errno).

→ Synopsis:

```
#include <pthread.h>
pthread_t   pthread_self(void);
```

Pthreads are refrenced by an ID of type 'pthread_t'. A thread can find out its id by calling this function

~void   * No errors are defined.
↳ID

→ Synopsis:

```
#include <pthread.h>
int pthread_create(
        pthread_t *restrict thread,
        const pthread_attr_t *restrict attr,
        void*(*start_routine)(void*),
        void* arg);
```

This function creates a Thread. ~~by~~

- thread: pointing to the ID of newly created thread.
- attr: represent an attribute object that encapsulate the attribute of a thread. If NULL, the new thread has default attributes.
- start-routine: the name of a function that the thread calls when it begin execution. It takes a single parameters specified by arg, a pointer to void. It return a pointer to void, which is treated as an exit status by pthread_join.

*S 0.

*US return nonzero error code.

→ Synopsis:

```
#include <pthread.h>
int pthread_join( pthread_t thread,
     void ** value-ptr);
```

This function susspends the calling thread untill the target thread, terminates.

~ thread: ID of terminating thread

~ value_ptr: provides a location for a pointer to the return status that the target thread passes to 'return'. or 'pthread- exit'.

*^{s} 0 *^{us} non-zero error code.

→ Synopsis:

```
#includ <pthread.h>
void pthread_exit(void* value-ptr);
```

A call to this system call causes only the calling thread to terminate. A thread that executes return from its top level implicitly calls this function.

~ value-ptr: this value is available to a successful 'pthread-join'.

* void. POSIX doe't define any

error for this function:

→ The process can terminat by calling 'exit' directly or by returning from main or by one of the other process threads call exit. In any of these case, all threads are terminated.

→ A call to 'exit' causes entire thread to terminate; a call to pthread_exit causes only the calling thread to terminate.

→ Parallelism is of two types
- Data level: in this type data is divided among multi-process/threads
- Task level: tasks are divided amon multi-process/threads.