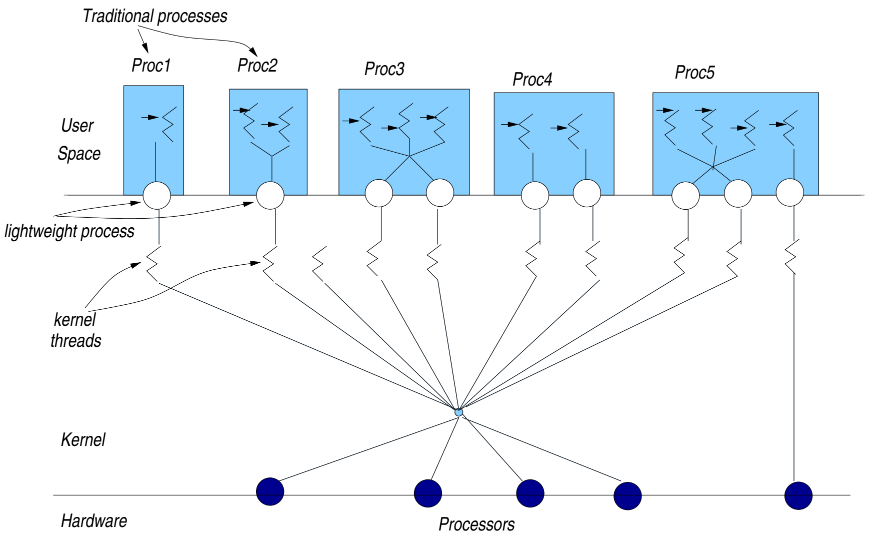
Threads

Spring 2025

Threads

- Threads are an alternative to multi-tasking.
- Try to overcome penalties when it comes to context switching and synchronization among different “flows” (or sequences) of execution.
- Offer a more efficient way to develop applications.

Thread (Solaris) Model



Thread Highlights

- One or more threads may be executed in the context of a process.
- The entity that is being scheduled is the thread – **not** the process itself.
- In the presence of a single processor, threads are executed concurrently.
- If there are more than one processors, threads can be assigned to different kernel thread (and so different CPUs) and run in parallel.
- Any thread may create a new thread.

Thread Highlights (continued)

- All threads of a single process share the same address space (address space, file descriptors etc.) BUT they have their own PC, stack and set of registers.
- Evidently, the kernel may manage *faster* the switch from one thread to another than the respective change from one process to another.
- The header `#include <pthread.h>` is required by all programs that use threads.
- Programs have to be compiled with the `pthread` library.
  `gcc <filename>.c -lpthread`

Thread Highlights (continued)

- The functions of the pthread library do not set the value of the variable errno and so, we cannot use the function perror() for the printing of a diagnostic message.
- If there is an error in one of the thread functions, strerror() is used for the printing of the diagnostic code (which is the "function return" for the thread).
- Function char *strerror(int errnum)
  - returns a pointer to a string that describes the error code passed in the argument errnum.
  - requires: #include <string.h>

Threads vs. Processes

	Threads	*Processes*
Address Space	Common. Any change made by one thread is visible to all (ie, `malloc()/free()`)	Different for each process Afer a `fork()` we have different address spaces
File Descriptors	Common. Any two threads can use the same descriptor One `close()` on this descriptor is sufficient	Two processes use copies of the file descriptors

What happens to threads when...

	What happens..
`fork`	Only the thread that invoked `fork` is duplicated.
`exit`	All threads die together (`pthread_exit` for the termination of a single thread).
`exec`	All threads disappear (the shared/common address space is replaced)
`signals`	This is somewhat more complex - Section 13.5 –*Robbins*[2] book

POSIX Thread Management

POSIX function	description
pthread_create	create a thread
pthread_self	find out own thread ID
pthread_equal	test 2 thread IDs for equality
pthread_exit	exit thread without existing process
pthread_detach	set thread to release resources
pthread_join	wait for a thread
pthread_cancel	terminate another thread
pthread_kill	send a signal to a thread

Creation of Threads

- The function that helps generate a thread is:

```
int pthread_create(pthread_t *thread,
         const pthread_attr_t *attr,
         void *(*start_routine) (void *), void *arg);
```

- creates a new thread with attributes specified by `attr` within a process.
- if `attr` is NULL, default attributes are used.
- Upon successful completion, `pthread_create()` shall store the ID of the created thread in the location referenced by *thread*.
- Through the `attr` we can change features of the thread but oftentimes we let the default value work, giving a `NULL`.
- If successful, the function returns 0; otherwise, an error number shall be returned to indicate the error.

Terminating a Thread

- `void pthread_exit(void *retval);`
- terminates the calling thread and makes the value `retvalue` available to any successful join with the terminating thread.
- After a thread has terminated, the result of access to local (auto) variables of the thread is undefined. So, references to local variables of the exiting thread should not be used for the `retvalue` parameter value.

pthread_join - waiting for thread termination

-
  ```
  int pthread_join(pthread_t thread, void **retval);
  ```
- suspends execution of the calling thread until the target `thread` terminates (unless the target `thread` *has already* terminated).
- When a `pthread_join()` returns successfully, the target `thread` has been terminated.
- On successful completion, the function returns 0.
- If retval is not NULL, then `pthread_join()` copies the exit status of the target thread into the location pointed to by `*retval`. If `thread` was canceled, then `PTHREAD_CANCELED` is placed in `*retval`.

Identifying - Detaching Threads

⟹ Get the calling thread-ID:

-
  ```
  pthread_t pthread_self(void);
  ```
- returns the thread-ID of the calling thread.

⟹ Detaching a thread:

-
  ```
  int pthread_detach(pthread_t thread);
  ```
- indicates that the storage for the `thread` can be reclaimed only when the thread terminates.
- If `thread` has not terminated, `pthread_detach()` shall not cause it to terminate.
- If the call succeeds, `pthread_detach()` shall return 0; otherwise, an error number shall be returned.
- Issuing a `pthread_join` on a detached thread fails.

Creating and using threads

```c
#include <stdio.h>
#include <string.h>     /* For strerror */
#include <stdlib.h>     /* For exit     */
#include <pthread.h>    /* For threads  */
#define perror2(s,e) fprintf(stderr, "%s: %s\n", s, strerror(e))

void *thread_f(void *argp){ /* Thread function */
   printf("I am the newly created thread %ld\n", pthread_self());
   pthread_exit((void *) 47); // Not recommended way of "exit"ing
   }                          // avoid using automatic variables
                              // use malloc-ed structs to return status
main(){
  pthread_t thr;
  int err, status;
  if (err = pthread_create(&thr, NULL, thread_f, NULL)) { /* New thread */
      perror2("pthread_create", err);
      exit(1);
      }
  printf("I am original thread %ld and I created thread %ld\n",
          pthread_self(), thr);
  if (err = pthread_join(thr, (void **) &status)) { /* Wait for thread */
      perror2("pthread_join", err); /* termination */
      exit(1);
      }
  printf("Thread %ld exited with code %d\n", thr, status);
  printf("Thread %ld just before exiting (Original)\n", pthread_self());
  pthread_exit(NULL);
  }
```

Outcome

```
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$ ./create_a_thread
I am original thread 140400641664832 and I created thread 140400633423616
I am the newly created thread 140400633423616
Thread 140400633423616 exited with code 47
Thread 140400641664832 just before exiting (Original)
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$
```

Using pthread_detach

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

#define perror2(s,e) fprintf(stderr,"%s: %s\n",s,strerror(e))

void *thread_f(void *argp){ /* Thread function */
   int err;
   if (err = pthread_detach(pthread_self())) {/* Detach thread */
       perror2("pthread_detach", err);
       exit(1);
       }
   printf("I am thread %ld and I was called with argument %d\n",
             pthread_self(), *(int *) argp);
   pthread_exit(NULL);
}
```

Using pthread_detach

```
main(){
   pthread_t thr;
   int err, arg = 29;

   if (err = pthread_create(&thr,NULL,thread_f,(void *) &arg)){/* New thread */
       perror2("pthread_create", err);
       exit(1);
       }
    printf("I am original thread %d and I created thread %d\n",
             pthread_self(), thr);
    pthread_exit(NULL);
}
```

→ Outcome:

```
antoulas@sazerac:~/Set007/src$ ./p16-detached_thread
I am thread 140411743098624 and I was called with argument 29
I will wait for some time in detached now.. 10 secs
I am original thread 140411751352064 and I created thread 140411743098624
I am the original and I am done!
About to leave detached thread now.. 140411743098624
antoulas@sazerac:~/Set007/src$
```

Create *n* threads that wait for random secs and then terminate

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
#define MAX_SLEEP 10 /* Maximum sleeping time in seconds */
#define perror2(s, e) fprintf(stderr, "%s: %s\n", s, strerror(e))

void *sleeping(void *arg) {
   long sl = (long) arg;
   printf("thread %ld sleeping %ld seconds ...\n", pthread_self(), sl);
   sleep(sl); /* Sleep a number of seconds */
   printf("thread %ld waking up\n", pthread_self());
   pthread_exit(NULL);
}

main(int argc, char *argv[]){
   int  n, i, err;
   long sl;
   pthread_t *tids;
   if (argc > 1) n = atoi(argv[1]); /* Make integer */
   else exit(0);
   if (n > 50) { /* Avoid too many threads */
      printf("Number of threads should be up to 50\n");
      exit(0);
      }

   if ((tids = malloc(n * sizeof(pthread_t))) == NULL) {
      perror("malloc");
      exit(1);
      }
```

n threads waiting for randm secs

```
   srandom((unsigned int) time(NULL)); /* Initialize generator */
   for (i=0 ; i<n ; i++) {
   sl = random() % MAX_SLEEP + 1; /* Sleeping time 1..MAX_SLEEP */
   if (err = pthread_create(tids+i, NULL, sleeping, (void *) sl)) {
      /* Create a thread */
      perror2("pthread_create", err);
      exit(1);
      }
   }

   for (i=0 ; i<n ; i++)
   if (err = pthread_join(*(tids+i), NULL)) {
      /* Wait for thread termination */
      perror2("pthread_join", err);
      exit(1);
      }
   printf("all %d threads have terminated\n", n);
}
```

Outcome

```
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$ ./random_sleeps 3
thread 140648973833984 sleeping 3 seconds ...
thread 140648965441280 sleeping 4 seconds ...
thread 140648982226688 sleeping 4 seconds ...
thread 140648973833984 waking up
thread 140648965441280 waking up
thread 140648982226688 waking up
all 3 threads have terminated
antoulas@sazerac:~/src$ ./random_sleeps 6
thread 140434098566912 sleeping 5 seconds ...
thread 140434115352320 sleeping 8 seconds ...
thread 140434081781504 sleeping 2 seconds ...
thread 140434090174208 sleeping 8 seconds ...
thread 140434106959616 sleeping 5 seconds ...
thread 140434073388800 sleeping 4 seconds ...
thread 140434081781504 waking up
thread 140434073388800 waking up
thread 140434098566912 waking up
thread 140434106959616 waking up
thread 140434115352320 waking up
thread 140434090174208 waking up
all 6 threads have terminated
antoulas@sazerac:~/src$
antoulas@sazerac:~/src$
```

Going from sigle- to multi-threaded programs

```c
#include <stdio.h>
#define NUM 5

void print_mesg(char *);

int     main(){
    print_mesg("hello");
    print_mesg("world\n");
}

void print_mesg(char *m){
int i;
for (i=0; i<NUM; i++){
    printf("%s", m);
    fflush(stdout);
    sleep(1);
    }
}
```

```
antoulas@sazerac:~/src$ ./print_single
hellohellohellohellohelloworld
world
world
world
world
antoulas@sazerac:~/src$
```

First Effort in Multi-threading

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

#define NUM 5

main()
{   pthread_t t1, t2;

    void *print_mesg(void *);

    pthread_create(&t1, NULL, print_mesg, (void *)"hello ");
    pthread_create(&t2, NULL, print_mesg, (void *)"world\n");
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
}

void *print_mesg(void *m)
{   char *cp = (char *)m;
    int i;

    for (i=0;i<NUM; i++){
        printf("%s", cp);
        fflush(stdout);
        sleep(2);
        }
    return NULL;
}
```

Outcome

```
antoulas@sazerac:~/src$ ./multi_hello
hello world
hello world
hello world
hello world
hello world
antoulas@sazerac:~/src$
```

What is “unexpected” here?

```
#include <stdio.h>
#include <pthread.h>
#define NUM  5
int      counter=0;

main(){
    pthread_t t1;
    void *print_count(void *);
    int i;

    pthread_create(&t1, NULL, print_count, NULL);
    for(i=0; i<NUM; i++){
        counter++;
        sleep(1);
    }
    pthread_join(t1,NULL);
}

void *print_count(void *m)
{
    /* counter is a shared variable */
    int i;
    for (i=0;i<NUM;i++){
        printf("count = %d\n",counter);
        sleep(1);
        /*changing this 1 -->> 0 has an effect */
    }
    return NULL;
}
```

The “unexpected” outcome:

```
antoulas@sazerac:~/src$ ./incprint
count = 1
count = 2
count = 3
count = 4
count = 5
antoulas@sazerac:~/src$
```

⊙ Changing *sleep(1)* ⟹ *sleep(0)*:

```
antoulas@sazerac:~/Set007/src$ vi incprint.c
antoulas@sazerac:~/src$ gcc incprint.c -o incprint -lpthread
antoulas@sazerac:~/src$ ./incprint
count = 1
count = 1
count = 1
count = 1
count = 1
antoulas@sazerac:~/Set007/src$
```

⇒ Reading may be inconsistent to what is written to `counter`:
Race Condition?

More problems!

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

int total_words;

int main(int ac, char *av[]){
    pthread_t t1, t2;
    void *count_words(void *);

    if (ac != 3 ) {
        printf("usage: %s file1 file2 \n", av[0]);
        exit(1);
    }
    total_words=0;
    pthread_create(&t1, NULL, count_words, (void *)av[1]);
    pthread_create(&t2, NULL, count_words, (void *)av[2]);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    printf("Main thread with ID: %ld reports %5d total words\n",
                pthread_self(), total_words);
}
```

More problems!

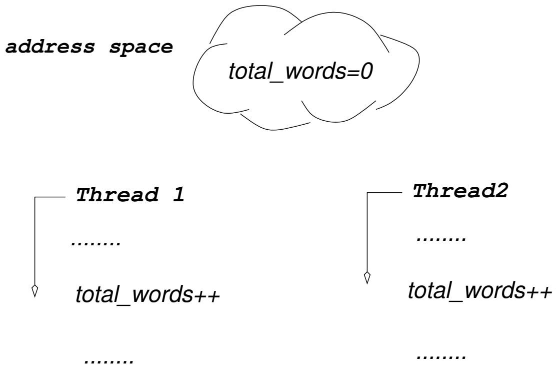
```c
void *count_words(void *f)
{
    char *filename = (char *)f;
    FILE *fp;    int c, prevc = '\0';
    printf("In thread with ID: %ld counting words.. \n",pthread_self());

    if ( (fp=fopen(filename,"r")) != NULL ){
        while ( ( c = getc(fp) )!= EOF ){
            if ( !isalnum(c) && isalnum(prevc) )
                total_words++;
            prevc = c;
            }
    fclose(fp);
    } else perror(filename);
    return NULL;
}
```

Outcome:

```
antoulas@sazerac:~/src$ !wc
wc -w fileA  fileB
 11136 fileA
  9421 fileB
 20557 total
antoulas@sazerac:~/src$ ./twordcount1 fileB fileA
In thread with ID: 140614526764800 counting words..
In thread with ID: 140614518372096 counting words..
Main thread with ID: 140614535006016 reports 20557 total words
antoulas@sazerac:~/src$ ./twordcount1 fileB fileA
In thread with ID: 140342756800256 counting words..
In thread with ID: 140342748407552 counting words..
Main thread with ID: 140342765041472 reports 20397 total words
antoulas@sazerac:~/src$ ./twordcount1 fileB fileA
In thread with ID: 140211362490112 counting words..
In thread with ID: 140211354097408 counting words..
Main thread with ID: 140211370731328 reports 20414 total words
antoulas@sazerac:~/src$ ./twordcount1 fileB fileA
In thread with ID: 140157487077120 counting words..
In thread with ID: 140157478684416 counting words..
Main thread with ID: 140157495318336 reports 20557 total words
antoulas@sazerac:~/src$ ./twordcount1 fileB fileA
In thread with ID: 139747725231872 counting words..
In thread with ID: 139747716839168 counting words..
Main thread with ID: 139747733473088 reports 20551 total words
antoulas@sazerac:~/src$
```

Race Condition



- *total_words* might NOT have a consistent value after executing the above two (concurrent) assignments.