Pthreads (POSIX Threads)

- a POSIX standard (IEEE 1003.1c) API for user thread creation and synchronization.
- API specifies behavior of the thread library, implementation is up to development of the library.
- Common in UNIX operating systems, but not in Windows.

Pthread commands

- The pthread_create function creates a new thread. It takes four arguments:
  - a thread variable or holder for the thread,
  - a thread attribute,
  - the function for the thread to call when it starts execution, and
  - an argument to the function. For example:

```
pthread_t          a_thread;
pthread_attr_t     a_thread_attribute;
void               thread_function(void *argument);
char               *some_argument;
pthread_create( &a_thread,
  a_thread_attribute,
  (void *)&thread_function,
  (void *)&some_argument);
```

From http://dis.cs.umass.edu/~wagner/threads_html/tutorial.html

Pthread Creation

- All Pthread programs must include the pthread.h header file
- `Pthread_t pid` declares the thread identifier
- A thread has a set of attributes set by the `pthread_attr_t attr` declaration, with `pthread_attr_init(&attr)`
- The thread itself is created with `pthread_create(&pid,&attr,&func,&args)` where func(args) is a function to be run in the new thread.

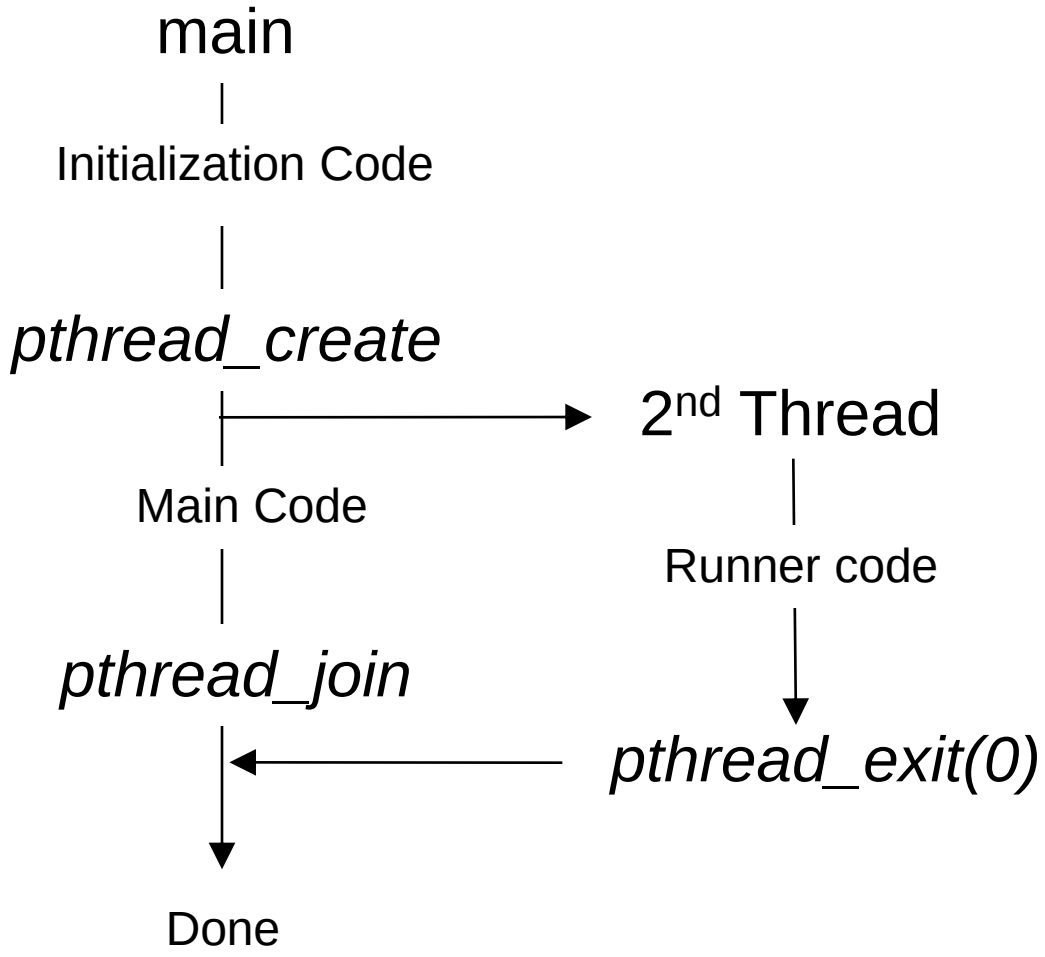
Pthread example code

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

int sum; /* this data is shared by the thread(s) */
void *runner(void *param); /* the thread */

main(int argc, char *argv[])
{
   pthread_t tid;       /* the thread identifier */
   pthread_attr_t attr; /* set of thread attributes */
   if (argc != 2) {
     fprintf(stderr, "usage: a.out <integer value>\n");
     exit();
   }
   if (atoi(argv[1]) < 0) {
     fprintf(stderr,"%d must be >= 0\n" ,atoi(argv[1])) ;
     exit();             /* atoi converts string to integer*/
   }
```

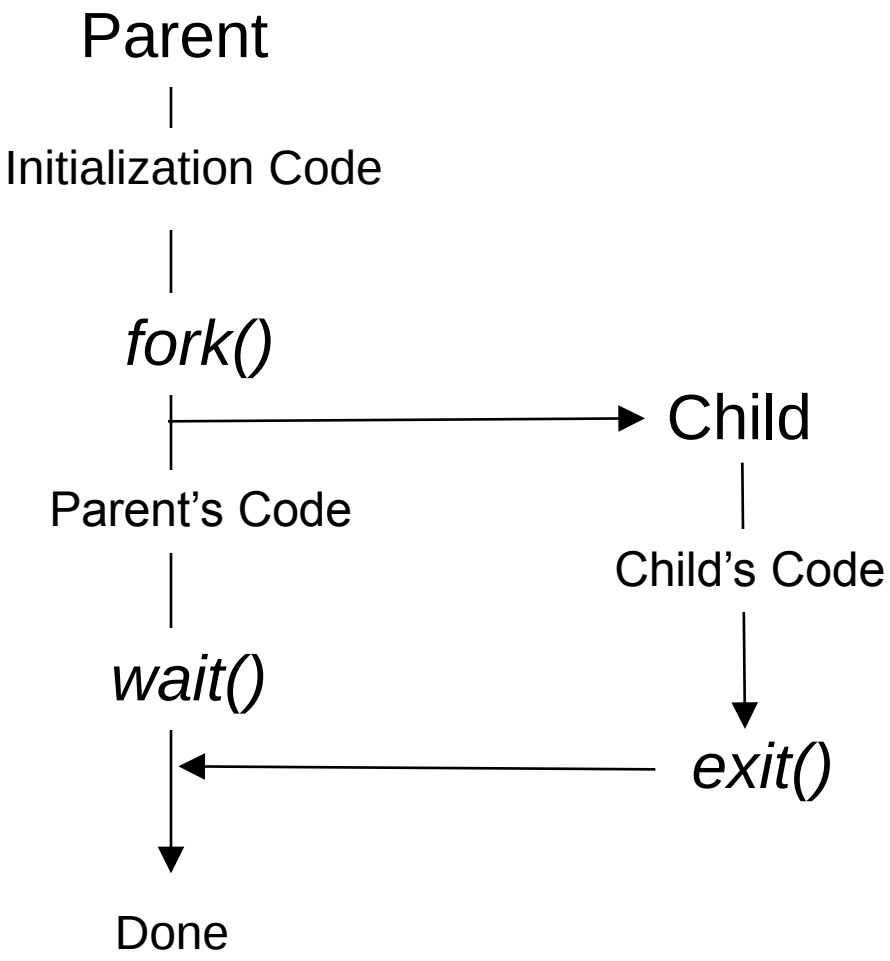
Pthread example code

```
    /* get the default attributes */
    pthread_attr_init(&attr) ;
    /* create the thread */
    pthread_create (&tid, &attr, runner, argv[1]);
    /* now wait for the thread to exit */
    pthread_join (tid, NULL) ;
    printf("sum = %d\n",sum);
}
/* The thread will begin control in this function */
void *runner(void *param)
{
    int upper = atoi(param);
    int i;
    sum = 0;
    if (upper > 0) {
      for (i = 1; i <= upper; i++)
        sum += i;
    }
pthread_exit(0);
}
```

The thread process



The fork process



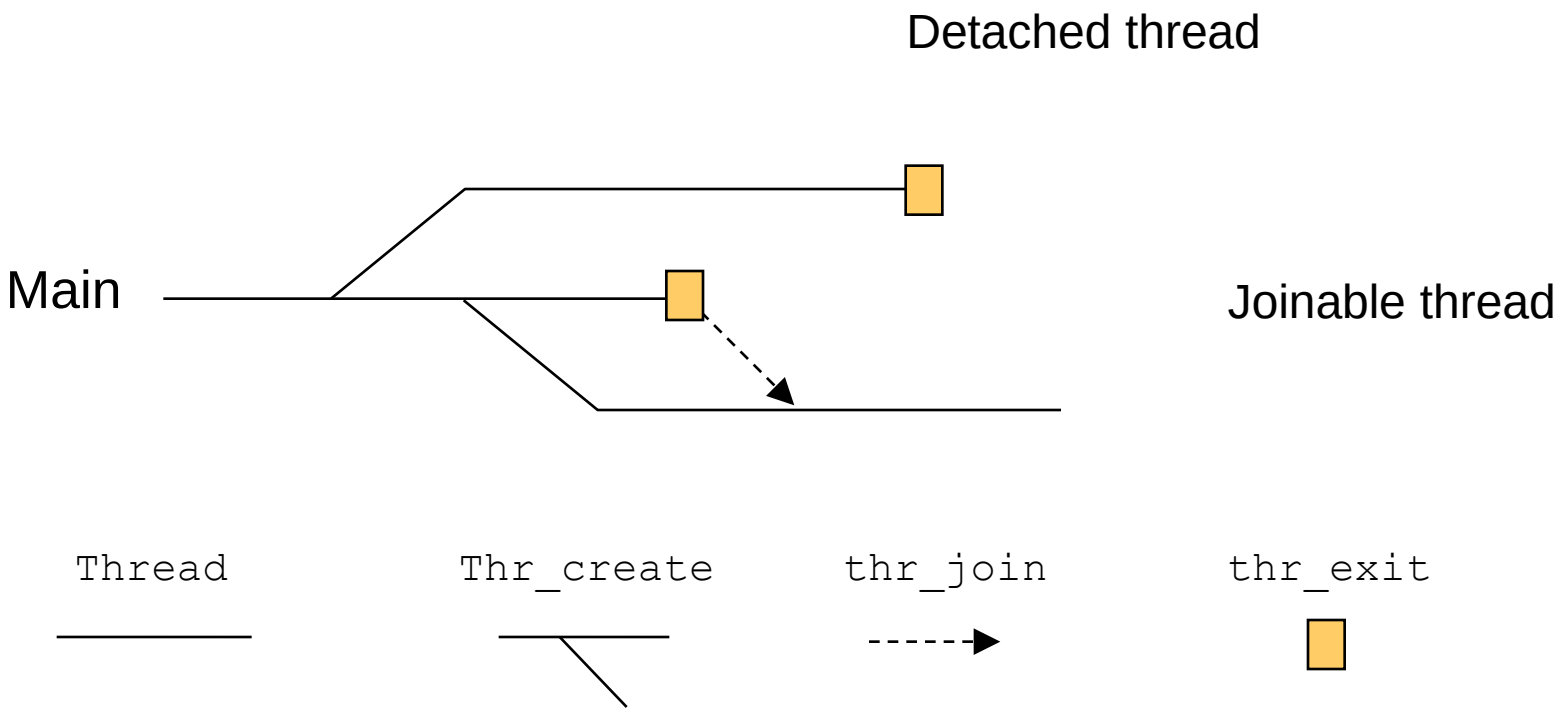
Thread Attributes

- Scope
  - The scope may be process-wide or system-wide
  - After creating an attribute object (eg. attr), the scope is set by `pthread_attr_setscope()`
  - If scope is defined `PTHREAD_SCOPE_SYSTEM`, then the thread is “bound” and the kernel can “see” the thread.
  - If scope is defined `PTHREAD_SCOPE_PROCESS`, then the thread is “unbound” and the kernel does not see the thread (default).

Thread Attributes

- Detach State
  - The detach state determines if the thread will be joinable from another thread.
  - After creating an attribute object (eg. attr), the detach state is set by `pthread_attr_setdetachstate()`
  - If detach state is defined `PTHREAD_CREATE_DETACHED`, then the thread is “detached” and the thread resources will be discarded on termination.
  - If detach state is defined `PTHREAD_CREATE_JOINABLE`, then the thread exit status and resources will be saved until the thread is joined by another thread (default).

Thread Creation and Joining



Thread Attributes

- Stack Address
  - The stack address specifies the base address of the stack for the thread.
  - After creating an attribute object (eg. attr), the stack address is set by `pthread_attr_setstackaddr()`
  - The stack address may be defined, but with care!
  - If stack address is defined `NULL`, then the system defines a stack address for the thread (default).

Thread Attributes

- Stack Size
  - The stack size specifies the size of the stack, in bytes, for the thread.
  - After creating an attribute object (eg. attr), the stack address is set by `pthread_attr_setstacksize()`
  - The stack address may be defined to a given value greater than `PTHREAD_STACK_MIN` bytes.
  - If stack size is defined `NULL`, then stack size is set to the system default (default).

Simple Thread Creation

Solaris

```
(void *) foo(int arg);
int arg;
thr_create(NULL, NULL, foo, (void *) arg, NULL, NULL);
```

pthread

```
(void *) foo(int arg);
int arg;
pthread_create(NULL, NULL, foo, (void *) arg);
```

Simple Thread Creation

OS/2

```
VOID foo(ULONG arg);
ULONG arg;
TID ThreadID;
ULONG stacksize = 0x500;
DosCreateThread(&ThreadID, (PFNTHREAD) foo, arg, NULL, stacksize);
```

Windows NT

```
DWORD foo(DWORD arg);
DWORD arg;
DWORD ThreadID;
CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE) foo,
 (LPVOID) arg, NULL, &ThreadID);
```

Solaris Threads and Pthreads

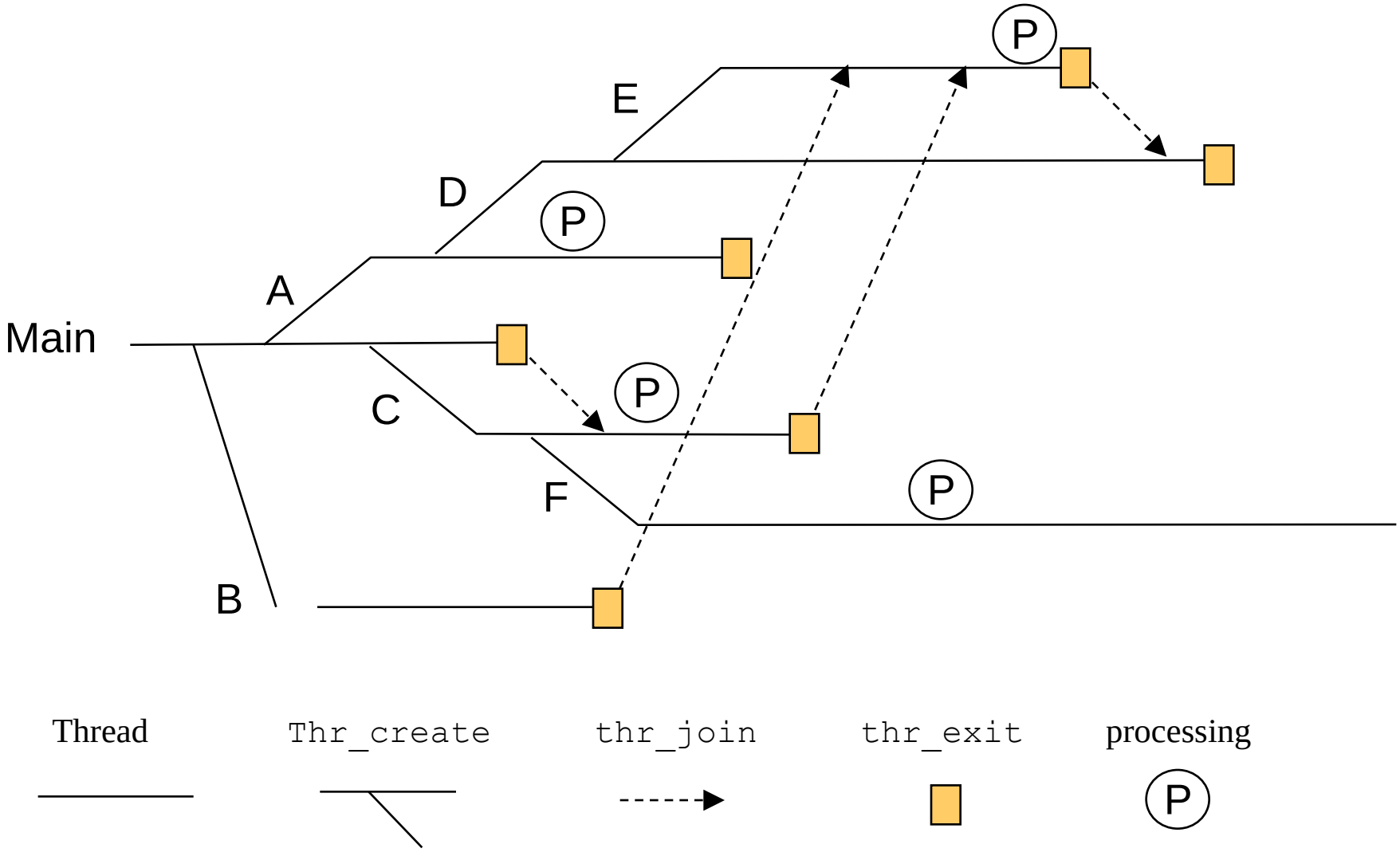
Solaris

- Reader/writer locks (many readers, single writer)
- Ability to suspend and continue a single thread
- Ability to create daemon threads
- Ability to set and get a level of concurrency

Pthreads

- Ability to cancel threads
- Attribute objects (thread and synchronization attributes)
- Scheduling policies

Thread Creation and Joining



Thread Creation and Joining

- There is no parent/child relationship between threads as there is for processes.
- Threads can be created and joined by many different threads in the process
- In the example
  - Main thread creates A, B and C, then exits
  - Thread B is created suspended
  - Main thread exits with *thr_exit()*, not *exit()* (which would have ended the whole process)
  - Main thread's exit status and resources are held until it is joined by thread C

Thread Create and Join Example

Variable initialization

```
#define _REENTRANT
#include <stdio.h>
#include <thread.h>

/ * Function prototypes for thread routines */
void *sub_a(void *);
void *sub_b(void *);
void *sub_c(void *);
void *sub_d(void *);
void *sub_e(void *);
void *sub_f(void *);

thread_t thr_a, thr_b, thr__c;
```

Thread Create and Join Example

Main thread

```
void main()
{
thread_t main_thr;

main_thr = thr_self(); /* returns thread ID for self */
printf("Main thread = %d\n", main_thr);

if (thr_create(NULL, 0, sub_b, NULL, THR_SUSPENDED|THR_NEW_LWP,
&thr__b))
    fprintf(stderr,"Can't create thr_b\n"), exit(1);

if (thr_create(NULL, 0, sub_a, (void *)thr_b, THR_HEW_LWP,
&thr_a))
    fprintf(stderr,"Can't create fchr_a\n"), exit(1);
```

Thread Create and Join Example

Main thread

```
if (thr_create(NULL, 0, sub_c,  (void *)main_thr, THR_NEW_LWP,
&thr_c))
   fprintf(stderr,"Can't create thr_c\n"), exit(1) ;

printf("Main Created threads A:%d B:%d C:%d\n", thr_a, thr_b,
thr_c) ;
   printf("Main Thread exiting...\n");

thr_exit((void *)main_thr) ;
}
```

Thread Create and Join Example

Thread A

```
void *sub_a(void *arg)
{
thread_t thr_b = (thread_t) arg;
thread_t thr_d;
int i;

printf("A: In thread A...\n");

if (thr_create(NULL, 0, sub_d, (void*)thr_b, THR_NEW_LWP,
&thr_d))
   fprintf(stderr, "Can't create thr_d\n"), exit(1);
```

Thread A immediately creates thread D

Thread Create and Join Example

Thread A

```
printf("A: Created thread D:%d\n", thr_d);

/* process
*/
for ( i=0; i<1000000*(int)thr_self 0; i++ ) ;

   printf("A: Thread exiting...\n") ;

thr_exit((void *)77) ;

}
```

On exit, Thread A resources are reclaimed by the OS, since it was created with the THR_DETACHED flag.

Thread Create and Join Example

Thread B

```
void * sub_b(void *arg)
{
int i;
printf("B: In thread B...\n");

/* process
*/

for ( i=0; i<1000000*(int)thr_self(); i++ ) ;
  printf("B: Thread exiting...\n") ;
thr_exit((void *)66);
}
```

Thread B was created suspended, so it runs only when thread D continues it with `thr_continue()`

Thread Create and Join Example

Thread C

```
void * sub__c(void *arg)
{
void *status;
int i;
thread_t main_thr, ret_thr;

main_thr - (thread_t)arg;

printf("C: In thread C...\n");

if (thr_create(NULL, 0, sub_f, (void *}0, THR_BOUND|THR_DAEMON,
NULL))
   fprintf(stderr, "Can't create thr_f\n"), exit(1);
```

Thread C creates thread F, then joins the main thread…

Thread Create and Join Example

Thread C

```
printf("C: Join main thread\n");

if (thr_join(main_thr,(thread_t *)&ret_thr, &status))
   fprintf(stderr, "thr_join Error\n"), exit(1);

printf("C: Main thread (%d) returned thread (%d) w/status
%d\n",
main_thr, ret_thr, (int) status);

/* simulated processing
*/
for ( i=0; i<1000000*(int)thr_self(); i++ ) ;

printf("C: Thread exiting...\n") ;
thr_exit((void *)88);
}
```

Thread Create and Join Example

Thread D

```
void * sub_d(void *arg)
{
thread_t thr_b = (thread_t) arg;
int i ;
thread_t thr_e, ret_thr;
void *status;

printf('D: In thread D...\n");

if (thr_create(NULL, 0, sub_e, NULL, THR_NEW_LWP, &thr_e))
fprintf(stderr,"Can'b create thr_e\n"), exit(1);

printf("D: Created thread E:%d\n", thr_e);
```

Thread D creates thread E,

Thread Create and Join Example

Thread D

```
printf("D: Continue B thread = %d\n", thr_b) ;
thr_continue(thr_b) ;

printf("D: Join E thread\n");

if(thr_join(thr_e,(thread_t *)&ret_thr, &status))
   fprintf(stderr,"thr_join Error\n"), exit(1);
```

Thread D continues thread B by making thr_continue() call, then tries to join thread E, blocking until thread E has exited.

Thread Create and Join Example

Thread D

```
printf("D: E thread (%d) returned thread (%d) w/status %d\n",
thr_e,
ret_thr, (int) status);

/* simulated processing
*/
for ( i=0; i<1000000 *(int)thr_self(); i++ );

printf("D: Thread exiting ...\n");
thr_exit((void *)55);
}
```

Thread D should be the last non-daemon thread running. When it exits, it should stop the daemon thread and stop execution of the process.

Thread Create and Join Example

Thread E

```
void * sub_e(void *arg)
{
int i ;
thread_t ret_thr;
void *status;

printf("E: In thread E...\n");

printf("E: Join A Chread\n");

if(thr_join(thr__a,(thread_t *)&ret_thr, &status))
   fprintf(stderr,"thr_join Error\n"), exit(1) ;

printf("E:A thread (%d) returned thread (%d) w/staCus %d\n",
ret_thr, ret_thr, (int) status);
```

Thread Create and Join Example

Thread E

```
printf("E: Join B thread\n");
if(thr_join(thr_b,(thread_t *)&ret_thr, &status))
 fprint f(stderr,"thr_join Error\n"), exit(11;

printf("E: B thread (%d) returned thread (%d) w/sLatus %d\n",
thr_b,
ret_thr, ( int) status) ;

printf("E: Join C thread\n" ) ;

if(thr_join(thr_c,(thread_t *)&ret_thr, & status))
   fprint f(stderr,"thr_join Error\n"), exit(1) ;
```

Thread E tries to join threads B and C, waiting for each of these threads to exit.

Thread Create and Join Example

Thread E

```
printf("E: C thread (%d) returned thread (%d) w/status %d\n" ,
thr_c,
ret_thr, (int) ytatus) ;

/* simulated processing
*/
for ( i=0; i<1000000*(int)thr_self(); i++);

printf("E: Thread exiting...\n"),
thr_exit((void *)-44);
}
```

Then thread E exits, holding its resources until joined by thread D.

Thread Create and Join Example

Thread F

```
 void *sub_f(void *arg)
{
int i;

printf("F: In thread F...\n");

while (1) {
    for (i=0,-i< 10000000,-i++) ;
    printf("F: Thread F is still running...\n") ;
    }
}
```

Thread F was created as a bound daemon thread, running on its own LWP until all the nondaemon threads have exited the process.

- useful for background processing