1. **Detach.c**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <signal.h>

#include <stdbool.h>

bool thread\_finished = 0;

void\* start\_function(void\* value)

{

printf("%s is now entering the thread function.\n", (char\*)value);

sleep(4);

thread\_finished =1;

printf("%s is now leaving the thread function.\n", (char\*)value);

pthread\_exit(value);

}

main()

{

int res,err;

pthread\_attr\_t attr;

pthread\_t thread1;

res = pthread\_attr\_init(&attr);

if (res != 0) {

perror("Attribute init failed");

exit(EXIT\_FAILURE);

}

res = pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_DETACHED);

if (res != 0) {

perror("Setting detached state failed");

exit(EXIT\_FAILURE);

}

res = pthread\_create(&thread1, &attr, start\_function,(void\*)"Thread1");

if (res != 0) {

perror("Creation of thread failed");

exit(EXIT\_FAILURE);

}

while(!thread\_finished) {

printf("Waiting for thread1 to finish.\n");

sleep(1);

}

printf("Child thread finished.\n");

pthread\_attr\_destroy(&attr);

}

1. **Scope.c**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#define NUM\_THREADS 5

void \*runner(void \*param);

main(int argc, char \*argv[])

{

int i, scope;

pthread\_t tid[NUM\_THREADS]; /\* the thread identifier \*/

pthread\_attr\_t attr; /\* set of attributes for the thread \*/

/\* get the default attributes \*/

pthread\_attr\_init(&attr);

/\* first inquire on the current scope \*/

if (pthread\_attr\_getscope(&attr,&scope) != 0)

fprintf(stderr, "Unable to get scheduling scope.\n");

else {

if (scope == PTHREAD\_SCOPE\_PROCESS)

printf("PTHREAD\_SCOPE\_PROCESS\n");

else if (scope == PTHREAD\_SCOPE\_SYSTEM)

printf("PTHREAD\_SCOPE\_SYSTEM\n");

else

fprintf(stderr,"Illegal scope value.\n");

}

/\* set the scheduling algorithm to PCS or SCS \*/

if (pthread\_attr\_setscope(&attr, PTHREAD\_SCOPE\_SYSTEM) != 0)

printf("unable to set scheduling policy.\n");

// we can't set PTHREAD\_SCOPE\_PROCESS

if (pthread\_attr\_setscope(&attr, PTHREAD\_SCOPE\_PROCESS) != 0)

printf("unable to set scheduling policy.\n");

/\* create the threads \*/

for (i = 0; i < NUM\_THREADS; i++)

pthread\_create(&tid[i],&attr,runner,NULL);

/\* Now join on each thread \*/

for (i = 0; i < NUM\_THREADS; i++)

pthread\_join(tid[i], NULL);

}

1. **Sched.c**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <pthread.h>

#include <string.h>

#include <sched.h>

void \*tf1(void \*arg);

void \*tf2(void \*arg);

void \*tf3(void \*arg);

char m1[] = " Thread1";

char m2[] = " Thread2";

char m3[] = " Thread3";

struct sched\_param schedparam1,schedparam2,schedparam3;

int main() {

int res,max\_priority,min\_priority;

pthread\_t t1,t2,t3;

pthread\_attr\_t at1,at2,at3;

schedparam1.sched\_priority = 40;

schedparam2.sched\_priority = 60;

schedparam3.sched\_priority = 80;

/\* Set attributes\*/

pthread\_attr\_init(&at1);

pthread\_attr\_setschedparam(&at1, &schedparam1);

pthread\_attr\_init(&at2);

pthread\_attr\_setschedparam(&at1, &schedparam1);

pthread\_attr\_init(&at3);

pthread\_attr\_setschedparam(&at1, &schedparam1);

max\_priority=sched\_get\_priority\_max(SCHED\_OTHER);

min\_priority=sched\_get\_priority\_min(SCHED\_OTHER);

printf(" MAX is %d and Min is %d\n",max\_priority,min\_priority);

printf("$$$$$$$$$$$$$$$$$$$$$$$$\n");

//Thread creatiom

res = pthread\_create(&t1, &at1, tf1, (void \*)m1);

res = pthread\_create(&t2, &at2, tf2, (void \*)m2);

res = pthread\_create(&t3, &at3, tf3, (void \*)m3);

pthread\_join(t1,NULL);

pthread\_join(t2,NULL);

pthread\_join(t3,NULL);

}

void \*tf1(void \*arg) {

int i;

printf("tf1 is running. Argument was %s\n", (char \*)arg);

for ( i=0;i<1000000000;i++);

strcpy(m1, "thread1 left");

puts(m1);

pthread\_exit(0);

}

void \*tf2(void \*arg) {

int i;

printf("tf2 is running. Argument was %s\n", (char \*)arg);

for (i=0;i<1000000000;i++);

strcpy(m1, "thread2 left");

puts(m1);

pthread\_exit("0");

}

void \*tf3(void \*arg) {

int i;

printf("tf3 is running. Argument was %s\n", (char \*)arg);

for (i=0;i<1000000000;i++);

strcpy(m1, "thread3 left");

puts(m1);

pthread\_exit(0);

}

1. **Cancel.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

void cleanup\_routine(void \*arg)

{

int \*c = (int\*)arg;

printf("ThreadCleanup: cleanup called at counter %d\n", \*c);

}

void \*threadFunc(void \*arg)

{

char \*str;

int i = 0;

int oldstate;

int retval;

pthread\_cleanup\_push(cleanup\_routine, &i);

pthread\_setcancelstate (PTHREAD\_CANCEL\_ENABLE, &oldstate);

str=(char\*)arg;

i = 0;

while(i < 110 )

{

usleep(1);

printf("threadFunc says: %s %d\n",str,i);

if ((i % 10)==0) {

pthread\_testcancel();

printf("pthread\_testcancel\n");

}

++i;

}

pthread\_cleanup\_pop(0);

return NULL;

}

int main(void)

{

pthread\_t pth; // this is our thread identifier

pthread\_attr\_t attr;

void \*result;

int status;

int join\_status;

int i = 0;

/\* Initialize and set thread detached attribute \*/

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);

pthread\_create(&pth,&attr,threadFunc,"foo");

while(i < 100)

{

usleep(1);

printf("main is running... %d\n",i);

if (i==20) {

printf("thread is terminating...\n");

status = pthread\_cancel(pth);

break;

}

++i;

}

printf("main waiting for thread to terminate...\n");

status = pthread\_join(pth,&result);

if (status != 0)

printf("Error: Join thread");

if (result == PTHREAD\_CANCELED)

printf ("Thread canceled at iteration\n");

else

printf ("Thread was not canceled\n");

printf("main with thread terminated\n");

return 0;

}