Федеральное агентство связи

Сибирский государственный университет телекоммуникаций и информатики

Лабораторная работа №8

Выполнил: студент группы ИП-211

Оганесян Альберт

Лацук Андрей

Проверил:

Профессор кафедры ПМиК

Малков Е. А.

Новосибирск 2024

**Задание:** протестируйте спин-блокировку используя фрагменты кода лекции 9.

**Дополнительное задание:** протестировать мьютекс блокировку используя фрагменты кода лекции 9. Сравнить время с спин-блокировкой.

**Цель:** знакомство с синхронизацией потоков.

1. Используем фрагменты кода из лекции 9, чтобы написать программу:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <signal.h>

#include <string.h>

#include <unistd.h>

volatile int running = 1;

char sh[6];

void \*Thread(void \*pParams);

void handle\_sigint(int sig)

{

running = 0;

}

int main(void)

{

pthread\_t thread\_id;

signal(SIGINT, handle\_sigint);

pthread\_create(&thread\_id, NULL, &Thread, NULL);

while (running)

{

printf("%s", sh);

fflush(stdout);

}

pthread\_cancel(thread\_id);

pthread\_join(thread\_id, NULL);

return 0;

}

void \*Thread(void \*pParams)

{

int counter = 0;

while (running)

{

if (counter % 2)

{

strcpy(sh, "Hello\n");

}

else

{

strcpy(sh, "Bye\_u\n");

}

counter++;

}

return NULL;

}

**2.** Запустим программу (Рис. 2.1). Основная проблема в том, что потоки работают одновременно, из-за чего основной поток по несколько раз выводит то же значение **sh**.

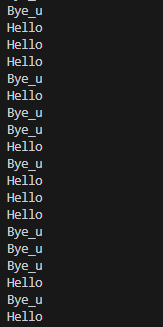


Рис. 2.1 Часть вывода программы

**3.** Теперь добавим в код спин-блокировку и алгоритм Питерсона, чтобы программа выводила обновленное значение только после измерения и добавим измерения времени:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <signal.h>

#include <string.h>

#include <unistd.h>

#include <sys/times.h>

#define MAX\_COUNT 100000

volatile int running = 1;

int turn = 1;

char sh[6];

pthread\_spinlock\_t spinlock;

void \*Thread(void \*pParams);

void handle\_sigint(int sig)

{

running = 0;

}

int main(void)

{

pthread\_t thread\_id;

struct tms start\_time, end\_time;

clock\_t real\_start\_time, real\_end\_time;

pthread\_spin\_init(&spinlock, PTHREAD\_PROCESS\_PRIVATE);

signal(SIGINT, handle\_sigint);

pthread\_create(&thread\_id, NULL, &Thread, NULL);

real\_start\_time = times(&start\_time);

while (running)

{

pthread\_spin\_lock(&spinlock);

if (turn == 0)

{

printf("%s", sh);

fflush(stdout);

turn = 1;

}

pthread\_spin\_unlock(&spinlock);

}

pthread\_cancel(thread\_id);

pthread\_join(thread\_id, NULL);

pthread\_spin\_destroy(&spinlock);

real\_end\_time = times(&end\_time);

double user\_time = (double)(end\_time.tms\_utime - start\_time.tms\_utime) / sysconf(\_SC\_CLK\_TCK);

double system\_time = (double)(end\_time.tms\_stime - start\_time.tms\_stime) / sysconf(\_SC\_CLK\_TCK);

printf("User CPU time: %.6f seconds\n", user\_time);

printf("System CPU time: %.6f seconds\n", system\_time);

return 0;

}

void \*Thread(void \*pParams)

{

int counter = 0;

while (counter < MAX\_COUNT)

{

pthread\_spin\_lock(&spinlock);

if (turn == 1)

{

if (counter % 2)

{

strcpy(sh, "Hello\n");

}

else

{

strcpy(sh, "Bye\_u\n");

}

counter++;

turn = 0;

}

pthread\_spin\_unlock(&spinlock);

}

running = 0;

return NULL;

}

**4.**Сделаем то же самое с Mutex блокировкой:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <signal.h>

#include <string.h>

#include <unistd.h>

#include <sys/times.h>

#define MAX\_COUNT 100000

volatile int running = 1;

int turn = 1;

char sh[6];

pthread\_mutex\_t mutex;

void \*Thread(void \*pParams);

void handle\_sigint(int sig)

{

running = 0;

}

int main(void)

{

pthread\_t thread\_id;

struct tms start\_time, end\_time;

clock\_t real\_start\_time, real\_end\_time;

pthread\_mutex\_init(&mutex, NULL);

signal(SIGINT, handle\_sigint);

pthread\_create(&thread\_id, NULL, &Thread, NULL);

real\_start\_time = times(&start\_time);

while (running)

{

pthread\_mutex\_lock(&mutex);

if (turn == 0)

{

printf("%s", sh);

fflush(stdout);

turn = 1;

}

pthread\_mutex\_unlock(&mutex);

}

pthread\_cancel(thread\_id);

pthread\_join(thread\_id, NULL);

pthread\_mutex\_destroy(&mutex);

real\_end\_time = times(&end\_time);

double user\_time = (double)(end\_time.tms\_utime - start\_time.tms\_utime) / sysconf(\_SC\_CLK\_TCK);

double system\_time = (double)(end\_time.tms\_stime - start\_time.tms\_stime) / sysconf(\_SC\_CLK\_TCK);

printf("User CPU time: %.6f seconds\n", user\_time);

printf("System CPU time: %.6f seconds\n", system\_time);

return 0;

}

void \*Thread(void \*pParams)

{

int counter = 0;

while (counter < MAX\_COUNT)

{

pthread\_mutex\_lock(&mutex);

if (turn == 1)

{

if (counter % 2)

{

strcpy(sh, "Hello\n");

}

else

{

strcpy(sh, "Bye\_u\n");

}

counter++;

turn = 0;

}

pthread\_mutex\_unlock(&mutex);

}

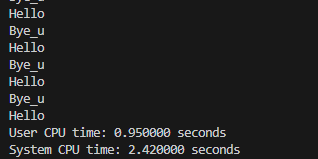
running = 0;

return NULL;

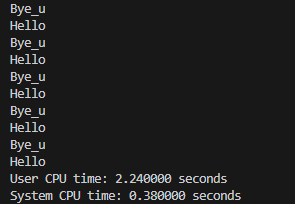
}

**5.** Запустим обе программы на значение counter до 100000 сравним время

Mutex:



Spin-lock:



**Вывод:** Мы научились работать с синхронизацией потоков и сравнили время выполнения mutex и spin-lock. Mutex затрачивает больше системного времени, а Spin-lock пользовательского.