Лабораторна робота №15

Тема: POSIX потоки (POSIX threads) та об'єкти синхронізації: мютекси (mutexes), умовні змінні (condition variables), семафори (semaphores), повідомлення (messages) в Unix-подібних операційних системах реального часу.
Мета роботи: навчитися програмувати POSIX потоки та об'єкти синхронізації, такі як: мютекси, умовні змінні, семафори, повідомлення в Unix-подібних операційних системах реального часу.

Теоретичні відомості

POSIX (***P****ortable* ***O****perating* ***S****ystem* ***I****nterface for Unix* – переносний інтерфейс операційних систем Unix) – набір стандартів, розроблених міжнародним інститутом електротехніки і електроніки (IEEE), що описують інтерфейси між операційною системою і прикладною програмою. POSIX потоки (або Pthreads) є визначені, як множина типів і функцій на мові C і реалізовані у header(pthread.h)/include файлах і бібліотеці потоків. Бібліотека потоків може бути частиною іншої бібліотеки, такої як libc. Наступні типи оголошенні у фалі #include <sys/types.h>.

```
pthread_t /* Використовується для ідентифікації потоку */
phread_attr_t /* Використовується для ідентифікації об'єкта атрибутів потоку */
size_t /* Використовується для збереження розмірів об'єктів. */
/* ф-ції ініціалізації та знищення об'єкту атрибутів потоку */
int pthread_attr_init(pthread_attr_t *);
int pthread_attr_destroy(pthread_attr_t *);
/* cancel execution of a thread */
int pthread_cancel(pthread_t);
/* detach a thread */
int pthread_detach(pthread_t);
/* compare thread IDs */
int pthread_equal(pthread_t, pthread_t);
/* thread termination */
void pthread_exit(void *);
/* wait for thread termination */
int pthread_join(pthread_t, void **);
/* get calling thread's ID */
pthread_t pthread_self(void);
/** Stack and scheduling related **/
/* set and get detachstate attribute */
int pthread_attr_setdetachstate(pthread_attr_t *, int);
int pthread_attr_getdetachstate(const pthread_attr_t *, int *);
/* set and get inheritsched attribute */
int pthread_attr_setinheritsched(pthread_attr_t *, int);
int pthread_attr_getinheritsched(const pthread_attr_t *, int *);
/* set and get schedparam attribute */
int pthread_attr_setschedparam(pthread_attr_t *, const struct sched_param *);
int pthread_attr_getschedparam(const pthread_attr_t *, struct sched_param *);
/* dynamic thread scheduling parameters access */
int pthread_getschedparam(pthread_t, int *, struct sched_param *);
int pthread_setschedparam(pthread_t, int , const struct sched_param *);
/* set and get schedpolicy attribute */
int pthread_attr_setschedpolicy(pthread_attr_t *, int);
```

```
int pthread_attr_getschedpolicy(const pthread_attr_t *, int *);
/* set and get stackaddr attribute */
int pthread_attr_setstackaddr(pthread_attr_t *, void *);
int pthread_attr_getstackaddr(const pthread_attr_t *, void **);
/* set and get stacksize attribute */
int pthread_attr_setstacksize(pthread_attr_t *, size_t);
int pthread_attr_getstacksize(const pthread_attr_t *, size_t *);
int pthread_getconcurrency(void);
void *pthread_getspecific(pthread_key_t);
```

Префікс функції	***Функціональна група***
pthread_	Потоки та інші різноманітні функції
pthread_attr_	Об’єкти атрибутів (властивостей) потоку
pthread_mutex_	Мютекси
pthread_mutexattr_	Об’єкти атрибутів (властивостей) мютекса
pthread_cond_	Умовні змінні
pthread_condattr_	Об’єкти умовних атрибутів
pthread_key_	Ключі потокових даних

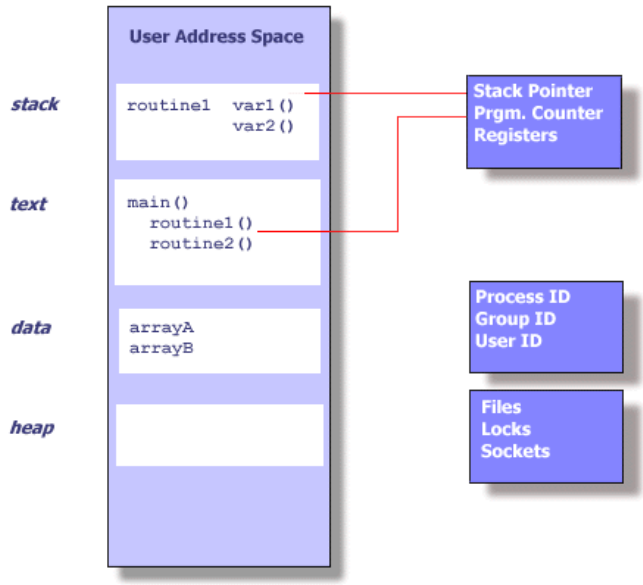

Рис.1 UNIX-процес

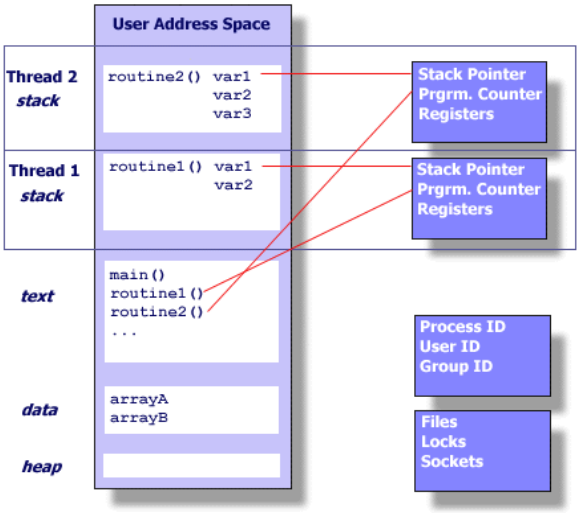

Рис.2 Потоки в одному процесі

Модель спільної пам’яті потоків (Рис.3):

- всі потоки мають спільну глобальну пам'ять;
- потоки мають власні дані;
- програміст несе відповідальність за синхронізований доступ до глобально-спільних даних.

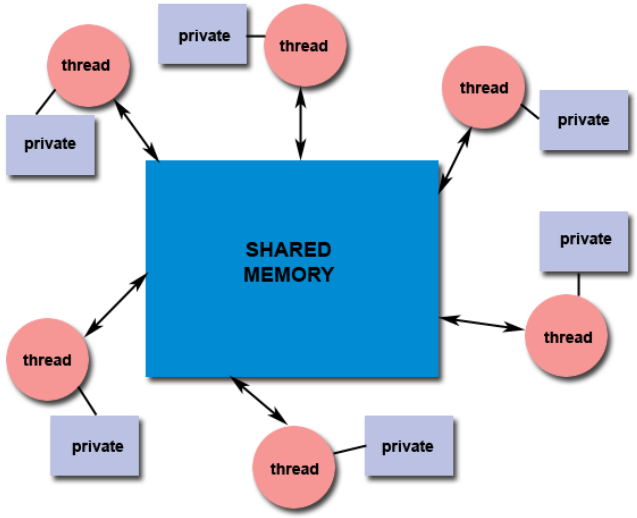

Рис.3 Модель спільної глобальної пам'яті

Максимальна кількість потоків, які можуть бути створенні процесом, залежить від самої реалізації. Створений потік може створити рівний собі потік (Рис.4). Не існує визначеної ієрархії або залежності між потоками.

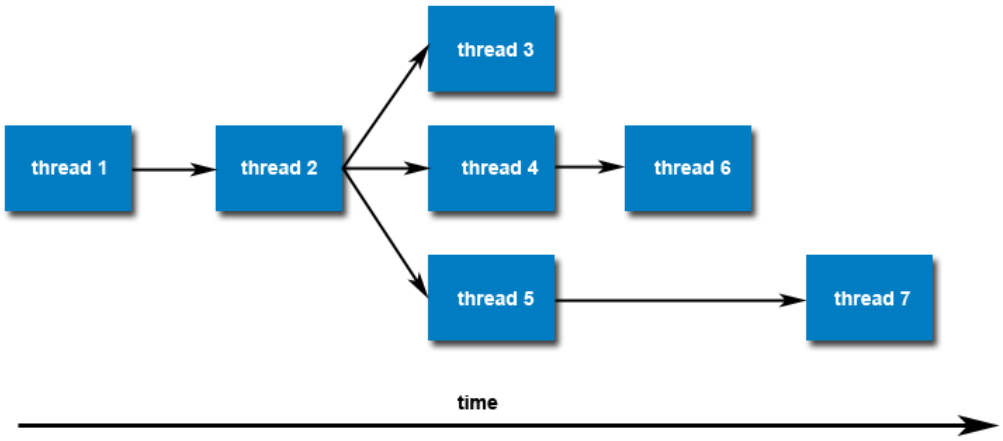

Рис.4 Створення потоків

Приклад програми "потоки":

```
/*
Complilation, linking and execution
$ gcc -c threads.c
$ gcc -lm -lrt -o threads threads.o
$ threads
*/

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

void message_printer_function(void *ptr)
{
        char *message;
        message =(char*)ptr;
        printf("%s \n",message);
```

```
}

int main(int argc, char **argv)
{
	pthread_t thread[5];
	pthread_attr_t attribute;
	int errorcode,counter,status;
	char *message="TestPrint";
	/* Initialize and set thread detached attribute */
	pthread_attr_init(&attribute);
	pthread_attr_setdetachstate(&attribute,PTHREAD_CREATE_JOINABLE);
	for(counter=0;counter<5;counter++)
	{
		printf("I am creating thread %d \n",counter);
		errorcode = pthread_create(&thread[counter],
		&attribute,(void*)&message_printer_function,(void*)message);
		if (errorcode)
		{
			printf("ERROR happened in thread creation");
			exit(-1);
		}
	}
	/* Free attribute and wait for the other threads */
	pthread_attr_destroy(&attribute);
	for(counter=0;counter<5;counter++)
	{
		errorcode = pthread_join(thread[counter],(void **)&status);
		if (errorcode)
		{
			printf("ERROR happened in thread join");
			exit(-1);
		}
		printf("Completed join with thread %d \n",counter);
		/*printf("Completed join with thread %d status=%d \n",counter,status);*/
	}
	pthread_exit(NULL);
}

/** POSIX Mutexes **/
/* Creating/Destroying Mutexes */
pthread_mutex_init(mutex, attr)
pthread_mutex_destroy(mutex)
pthread_mutexattr_init(attr)
pthread_mutexattr_destroy(attr)
/* Locking/Unlocking Mutexes */
pthread_mutex_lock(mutex)
pthread_mutex_trylock(mutex)
pthread_mutex_unlock(mutex)

/** POSIX Condition Variables **/
/* Creating/Destroying Condition Variables */
```

```
pthread_cond_init(condition, attr)
pthread_cond_destroy(condition)
pthread_condattr_init(attr)
pthread_condattr_destroy(attr)
/* Waiting/Signalling On Condition Variables */
pthread_cond_wait(condition, mutex)
pthread_cond_signal(condition)
pthread_cond_broadcast(condition)
```

Приклад програми "мютекси":

```
/*
Compilation, linking and execution
$ gcc -c mutexes.c
$ gcc -lm -lrt -o mutexes mutexes.o
$ mutexes
*/

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

#define SET 1
#define NOTSET 0

int info_in_buffer=NOTSET;
pthread_mutex_t lock=PTHREAD_MUTEX_INITIALIZER;

void read_user(void)
{
	while(1)
	{
		/* check whether buffer is written and read data */
		pthread_mutex_lock(&lock);
		if (info_in_buffer==SET)
		{
			printf("In read user \n");
			/* simulation the read operation by a wait (sleep(2)) */
			sleep(2);
			info_in_buffer=NOTSET;
		}
		pthread_mutex_unlock(&lock);
		/* giving the writer an opportunity to write to the buffer */
		sleep(2);
	}
}

void write_user(void)
{
	while(1)
	{
		/* check whether buffer is free and write data */
		pthread_mutex_lock(&lock);
```

```
                if (info_in_buffer==NOTSET)
                {
                        printf("In write user \n");
                        /* simulation the write operation by a wait (sleep(2)) */
                        sleep(2);
                        info_in_buffer=SET;
                }
                pthread_mutex_unlock(&lock);
                /* giving the reader an opportunity to read from the buffer */
                sleep(2);
        }
}

int main(int argc, char **argv)
{
        pthread_t Readthread;
        pthread_attr_t attribute;
        pthread_attr_init(&attribute);
        pthread_create(&Readthread,&attribute,(void*)&read_user,NULL);
        write_user();
}
```

POSIX семафори

В POSIX є семафори-лічильники, бінарні семафори, що дозволяють процесам, запущених в різних адресних просторах, або потокам запущених в спільному адресному просторі, синхронізуватися і взаємодіяти використовуючи спільну пам'ять. Наступний набір функцій використовується для роботи з семафорами:

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
/* Initializes the semaphore object pointed by ‘sem’ */
int sem_destroy(sem_t *sem);
/* Destroys a semaphore object and frees up the resources it might hold */
/* The following three functions are used in conjunction with other processes. See man pages for
more details. */
sem_t *sem_open(const char *name, int oflag, ...);
int sem_close(sem_t *sem);
int sem_unlink(const char *name);
int sem_wait(sem_t *sem);
/* Suspends the calling thread until the semaphore pointed to by ‘sem’ has
non-zero count. Decreases the semaphore count. */
int sem_trywait(sem_t *sem);
/* A non-blocking variant of sem_wait. */
int sem_post(sem_t *sem);
/* Increases the count of the semaphore pointed to by ‘sem’. */
int sem_getvalue(sem_t *sem, int *sval);
/* Stores the current count of the semaphore ‘sem’ in ‘sval’. */
```

Приклад програми “семафори”:

```
/*
Complilation, linking and execution
```

```
$ gcc -c semaphores.c
$ gcc -lm -lrt -o semaphores semaphores.o
$ semaphores
*/

#include<stdio.h>
#include<pthread.h>
#include<semaphore.h>
#include<sys/sem.h>

sem_t writer_lock;
sem_t reader_lock;

void read_user(void)
{
        while(1)
        {
                sem_wait(&reader_lock);
                /* simulate read operation by a delay */
                printf("in reader task \n");
                sleep(2);
                sem_post(&writer_lock);
        }
}

void write_user(void)
{
        while(1)
        {
                sem_wait(&writer_lock);
                /* simulate read operation by a delay */
                printf("in writer task \n");
                sleep(2);
                sem_post(&reader_lock);
        }
}

int main(int argc, char **argv)
{
        pthread_t read_thread;
        pthread_attr_t attribute;
        sem_init(&writer_lock,0,1);
        sem_init(&reader_lock,0,1);
        sem_wait(&reader_lock);
        pthread_attr_init(&attribute);
        pthread_create(&read_thread,&attribute,(void*)&read_user,NULL);
        write_user();
}

mqd_t mq_open(const char *name, int oflag, ...);
/* Connects to, and optionally creates, a named message queue. */
```

```
int mq_send(mqd_t mqdes, const char *msg_ptr, oskit_size_t msg_len,
unsigned int msg_prio);
/* Places a message in the queue. */
int mq_receive(mqd_t mqdes, char *msg_ptr, oskit_size_t msg_len,
unsigned int *msg_prio);
/* Receives (removes) the oldest, highest priority message from the queue. */
int mq_close(mqd_t mqdes);
/* Ends the connection to an open message queue. */
int mq_unlink(const char *name);
/* Ends the connection to an open message queue and causes the
queue to be removed when the last process closes it. */
int mq_setattr(mqd_t mqdes, const struct mq_attr *mqstat, struct mq_attr *omqstat);
int mq_getattr(mqd_t mqdes, struct mq_attr *mqstat);
/* Set or get message queue attributes. */
int mq_notify(mqd_t mqdes, const struct sigevent *notification);
/* Notifies a process or thread that a message is available in the queue. */
```

POSIX повідомлення

Черги повідомлень працюють обмінюючись даними через буфери. Будь-яка кількість процесів може взаємодіяти через черги повідомлень. Повідомлення може бути як синхронним так і асинхронним. Наступний приклад ілюструє роботу з повідомленнями:

Приклад програми "повідомлення":

```
/*
Compilation, linking and execution
$ gcc -c message_send.c
$ gcc -lm -lrt -o message_send message_send.o
$ message_send
*/

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>
#include <string.h>

#define MSGSZ 128

/* Declare the message structure. */
typedef struct msgbuf
{
        long mtype;
        char mtext [MSGSZ];
}       message_buf;

int main(int argc, char **argv)
{
        int msqid;
        int msgflg = IPC_CREAT | 0666;
        key_t key;
```

```
        message_buf sbuf;
        size_t buf_length;
        /* Get the message queue id for the "name"1234, which was created by the server. */
        key = 1234;
        (void)fprintf(stderr,"\nmsgget:Calling msgget(%#lx, \%#o) \n",key,msgflg);
        if ((msqid = msgget(key,msgflg))<0)
        {
                perror("msgget");
                exit(1);
        }
        else
                (void)fprintf(stderr,"msgget:msgget succeeded:msqid =%d \n",msqid);
        /* We'll send message type 1 */
        sbuf.mtype = 1;
        (void)fprintf(stderr,"msgget:msgget succeeded:msqid =%d \n",msqid);
        (void)strcpy(sbuf.mtext,"Did you get this stuff?");
        (void)fprintf(stderr,"msgget:msgget succeeded:msqid =%d \n",msqid);
        buf_length = strlen(sbuf.mtext)+1 ;
        /* Send a message. */
        if (msgsnd(msqid,&sbuf,buf_length,IPC_NOWAIT)<0)
        {
                printf ("%d,%d,%s,%d \n",msqid,sbuf.mtype,sbuf.mtext,buf_length);
                perror("msgsnd");
                exit(1);
        }
        else
                printf("Message: \"%s\" Sent \n",sbuf.mtext);
        exit(0);
}

/*
Compilation, linking and execution
$ gcc -c message_rec.c
$ gcc -lm -lrt -o message_rec message_rec.o
$ message_rec
*/

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>
#include <stdio.h>

#define MSGSZ 128

/* Declare the message structure. */
typedef struct msgbuf
{
        long mtype;
        char mtext[MSGSZ];
} message_buf;
```

```
int main(int argc, char **argv)
{
        int msqid;
        key_t key;
        message_buf rbuf;
        /* Get the message queue id for the "name"1234,which was created by the server.*/
        key = 1234;
        if ((msqid =msgget(key, 0666))<0)
        {
                perror("msgget");
                exit(1);
        }
        /* Receive an answer of message type 1. */
        if (msgrcv(msqid,&rbuf,MSGSZ,1,0)<0)
        {
                perror("msgrcv");
                exit(1);
        }
        /* Print the answer.*/
        printf("%s \n",rbuf.mtext);
        exit(0);
}
```

3. Завдання.

Модифікувати вище описані програми використовуючи механізми синхронізації для зчитування інформації з файлу (текстовий або бінарний файл) і її виводу через спільний буфер обміну даними.

4. Зміст звіту

1. Титульна сторінка та тема роботи.
2. Мета роботи.
3. Текст програм і протокол їх роботи в системі.
4. Висновки.

5. Контрольні запитання

1. Назвіть механізми синхронізації в багатозадачних ОС РЧ.
2. Що таке процес (process) ?
3. Що таке “потік” (thread) ? Функції роботи з потоками в POSIX-сумісних ОС РЧ.
4. Яка відмінність між процесом і потоком ?
5. Що таке “мютекс” ? Функції роботи з мютексами в POSIX-сумісних ОС РЧ.
6. Що таке “семафор” ? Функції роботи з семафорами в POSIX-сумісних ОС РЧ.
7. Що таке “повідомлення” (message), черга повідомлень (message queue) ?. Функції роботи з його суть в ОС UNIX.