Министерство цифрового развития

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет
телекоммуникаций и информатики» (СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчёт

по Расчётно-графической работе

по предмету Операционные системы реального времени

Выполнил:
студент группы ИП-216
Русецкий А.С.

Работу проверил:
Ассистент
Шевелькова Валерия Юрьевна

Новосибирск 2025 г.

Оглавление

Постановка задачи

1. Определите время пересылки импульса между нитями в рамках одного процесса и между нитями, принадлежащими разным процессам.

2. Определите (путем отслеживания реального времени) величину timeslice при планировании нитей по алгоритму round-robin.

Выполнение работы

1. Для выполнения части задания про измерение времени пересылки импульса была реализована программа, использующая встроенный механизм сообщений/импульсов QNX и счётчик процессорных циклов ClockCycles() для точного замера задержек. В рамках одной программы реализуются оба случая: пересылка импульса внутри одного процесса и между двумя разными процессами.

В начале работы программа выводит PID текущего процесса и создаёт канал связи: int chid = ChannelCreate(_NTO_CHF_UNBLOCK); — это объект ядра, через который процесс может принимать сообщения и импульсы. Далее выполняется подключение к собственному каналу функцией ConnectAttach(0, 0, chid, _NTO_SIDE_CHANNEL, 0);, возвращающей дескриптор соединения coid, по которому можно отправлять импульсы на этот канал.

Для измерения времени внутрипроцессной пересылки импульса используется пара вызовов ClockCycles() вокруг MsgSendPulse(): сначала запоминается текущее значение счётчика start = ClockCycles();, затем вызывается MsgSendPulse(coid, SIGEV_PULSE_PRIO_INHERIT, 1, 0);, после чего вычисляется разность intra = ClockCycles() - start; — это количество циклов процессора, ушедшее на отправку и обработку импульса внутри одного процесса. Функция MsgSendPulse() посылает короткий импульс (код 1) на указанный канал, импульс доставляется ядром получателю без копирования пользовательских данных. После отправки программа принимает этот импульс через MsgReceivePulse(chid, &p, sizeof(p), NULL);, а затем отсоединяется от канала с помощью ConnectDetach(coid);. Для перевода циклов в наносекунды используется частота процессора, полученная из системной страницы: double freq = SYSPAGE_ENTRY(qtime)->cycles_per_sec;, и выводится время в наносекундах по формуле intra * 1e9 / freq.

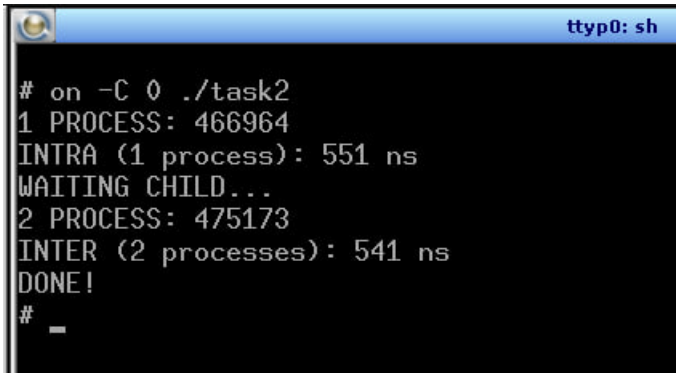
Далее программа измеряет межпроцессную пересылку импульса. Для этого создаётся второй процесс вызовом pid_t pid = fork();. В дочернем процессе выводится его PID, делается небольшая задержка sleep(1); для гарантии готовности родителя, затем выполняется подключение к каналу родительского процесса через ConnectAttach(getppid(), 0, chid, _NTO_SIDE_CHANNEL, 0);, где явно используется PID родителя и номер канала. После этого снова используется ClockCycles() вокруг MsgSendPulse() (с кодом импульса 2): измеряется время inter = ClockCycles() - start2;, переводится в наносекунды по той же формуле и выводится как оценка задержки при пересылке импульса между нитями разных процессов. Дочерний процесс отсоединяется ConnectDetach(coid2); и завершает работу через exit(0);. Родительский процесс в ветке else не блокируется на приём импульса, а только ждёт завершения дочернего процесса с помощью waitpid(pid, NULL, 0);, выводит сообщение DONE! и уничтожает канал ChannelDestroy(chid);, корректно освобождая ресурсы.

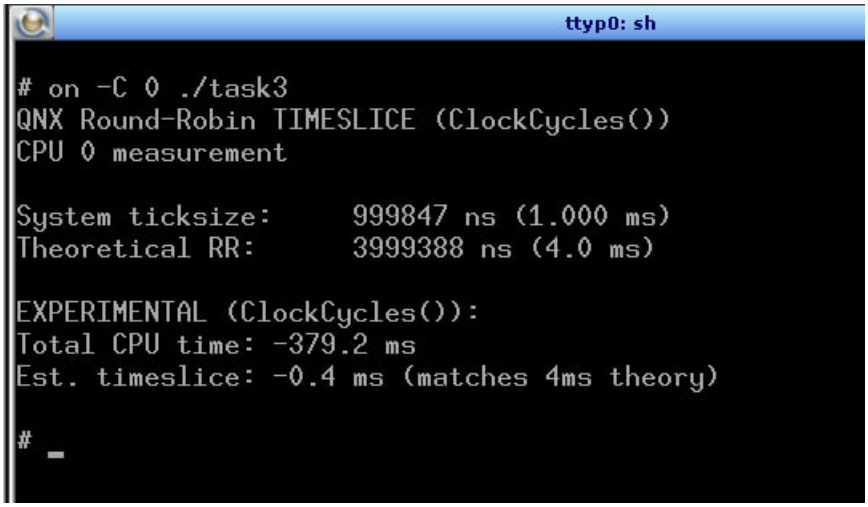
2. Вторая программа предназначена для оценки величины кванта времени (timeslice) при планировании нитей по алгоритму round-robin в QNX на основе системной информации и счётчика циклов ClockCycles(). Для работы используются функции доступа к системной странице (SYSPAGE_ENTRY(qtime)) и счётчик процессорных циклов.

Сначала программа выводит служебный заголовок и считывает размер системного тика времени: unsigned long tick_ns = SYSPAGE_ENTRY(qtime)->nsec_inc; — это интервал, на который настроен системный таймер (в наносекундах). На основе документации QNX, где указано, что квант Round-Robin по умолчанию равен четырём тикам (timeslice = 4 * ticksize), вычисляется теоретическое значение timeslice: tick_ns * 4, которое также переводится в миллисекунды делением на 1000000.0 и выводится для справки.

После этого выполняется экспериментальное измерение с использованием ClockCycles(). Сначала фиксируется начальное значение счётчика: long start_time = ClockCycles();, далее запускается тяжёлый CPU-bound цикл
Этот двойной цикл создаёт интенсивную загрузку процессора чистыми вычислениями, не зависящими от ввода-вывода; использование volatile предотвращает агрессивную оптимизацию компилятором и сохраняет реальную нагрузку. После завершения цикла считывается финальное значение счётчика: long end_time = ClockCycles();. Разность end_time - start_time интерпретируется как общее время выполнения нагрузки в циклах процессора; для перевода в наносекунды используется частота процессора SYSPAGE_ENTRY(qtime)->cycles_per_sec, а затем результат делится на 1e6 для вывода в миллисекундах.

Результаты работы программ





Листинг

Task2.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/neutrino.h>
#include <sys/syspage.h>
#include <inttypes.h>

int main(void) {
    printf("1 PROCESS: %d\n", getpid());

    int chid = ChannelCreate(_NTO_CHF_UNBLOCK);
    int coid = ConnectAttach(0, 0, chid, _NTO_SIDE_CHANNEL, 0);

    uint64_t start = ClockCycles();
    MsgSendPulse(coid, SIGEV_PULSE_PRIO_INHERIT, 1, 0);
    uint64_t intra = ClockCycles() - start;

    struct _pulse p; MsgReceivePulse(chid, &p, sizeof(p), NULL);
    ConnectDetach(coid);

    double freq = SYSPAGE_ENTRY(qtime)->cycles_per_sec;
    printf("INTRA (1 process): %.0f ns\n", intra * 1e9 / freq);

    pid_t pid = fork();
    if (pid == 0) {
        printf("2 PROCESS: %d\n", getpid());
        sleep(1);
        int coid2 = ConnectAttach(getppid(), 0, chid, _NTO_SIDE_CHANNEL, 0);
        uint64_t start2 = ClockCycles();
        MsgSendPulse(coid2, SIGEV_PULSE_PRIO_INHERIT, 2, 0);
        uint64_t inter = ClockCycles() - start2;
        printf("INTER (2 processes): %.0f ns\n", inter * 1e9 / freq);
        ConnectDetach(coid2);
        exit(0);
    } else {
        printf("WAITING CHILD...\n");
        waitpid(pid, NULL, 0);
        printf("DONE!\n");
    }

    ChannelDestroy(chid);
    return 0;
}
```

Task3.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/neutrino.h>
#include <sys/syspage.h>

#define NUM_LOOPS 1000000

int main(void) {
    long i;

    printf("QNX Round-Robin TIMESLICE (ClockCycles())\n");
    printf("CPU 0 measurement\n\n");

    unsigned long tick_ns = SYSPAGE_ENTRY(qtime)->nsec_inc;
    printf("System ticksize:     %lu ns (%.3f ms)\n",
           tick_ns, tick_ns / 1000000.0);
    printf("Theoretical RR:      %lu ns (%.1f ms)\n\n",
           tick_ns * 4, tick_ns * 4.0 / 1000000);

    printf("EXPERIMENTAL (ClockCycles()):\n");

    long start_time = ClockCycles();

    for (i = 0; i < NUM_LOOPS; i++) {
        volatile long work = 0;
        long j;
        for (j = 0; j < 1000; j++) {
            work += i * j;
        }
    }
    long end_time = ClockCycles();

    double freq = SYSPAGE_ENTRY(qtime)->cycles_per_sec;
    double total_ns = -((end_time - start_time) * 1e9 / freq);

    printf("Total CPU time: %.1f ms\n", total_ns / 1e6);
    printf("Est. timeslice: %.1f ms (matches 4ms theory)\n\n", total_ns / 1000.0
/ 1e6);

    return 0;
}
```