

Федеральное агентство связи  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Сибирский государственный университет телекоммуникаций и  
информатики»

## **Расчетно-Графическое Задание**

Выполнил: студент 4 курса

ИВТ, гр. ИП-713

Михеев Н.А.

Проверил: старший преподаватель кафедры ПМиК

Милешко А.В.

Новосибирск, 2020 г.

## **Оглавление**

<b>1. Задание .....</b>	<b>3</b>
<b>2. Выполнение задания .....</b>	<b>3</b>
<b>Часть №1 .....</b>	<b>3</b>
<b>Часть №2 .....</b>	<b>4</b>
<b>3. Листинг .....</b>	<b>5</b>

## 1. Задание

1. Исследуйте эффективность по времени выделения буфера в стеке (с помощью `alloca()`) и в куче (с помощью `malloc()`).
2. Не все системы позволяют передавать сигналы и прерывания по сети. Проверьте экспериментально, работает ли доставка событий типа сигнала и прерывания в сети.

## 2. Выполнение задания

### Часть №1

Для выполнения первого задания была реализована программа с использованием функций `alloca`, `malloc`, для отслеживания времени был выбран `ClockCycles()`. Все выделения буфера, что в стеке, что в куче проведены по 100 раз и взято среднее время. Далее о каждой использованной функции подробнее:

- `alloca(size_t size)` – функция из библиотеки `malloc.h`, выделяет `size` байт памяти из стека системы и возвращается указатель на него. Если запрос на выделение памяти не был выполнен успешно, то возвращается нулевой указатель и дальнейшее поведение программы не определено. Стоит отметить, что при использовании `alloca()` память освобождается автоматически при выходе из функции, которая вызвала `alloca()`;
- `malloc(size_t size)` – функция из библиотеки `malloc.h`, возвращается адрес первого байта области памяти размера `size` байт, которая была выделена из кучи. Если памяти недостаточно и выделение не было произведено успешно, то возвращается нулевой указатель. Попытка использования такого нулевого указателя может привести к падению системы.
- `ClockCycles()` – функция из библиотеки `sys/neutrino.h`, возвращает текущее значение 64-битного счетчика циклов процессора. Чтобы вычислить количество прошедших секунд необходимо сначала вычислить количество циклов в секунду у системы с помощью `SYSPAGE_ENTRY(qtime)->cycles_per_sec` и уже поделить наши циклы на это количество.

Замеры были произведены на выделении буфера размером 100 байт, преимущество `alloca` по количеству произведенных циклов ~40%, по времени различие уже не так заметно «на глаз», но все же 2.7 мкс у `alloca` против 4.7 мкс у `malloc`.

Хоть `alloca` занимает очень мало места и очень быстрый, но при ошибках аллокации памяти поведение его не определено. Так что, при выделении больше чем несколько сотен байт памяти, лучше использовать `malloc`, при его использовании мы все равно можем получить ошибки выделения памяти, но хотя бы мы увидим какую-то индикацию самой ошибки вместо простого «уничтожения» (как это на западе называют `blowing off`) стека.

Результаты работы программы на скриншоте:

```
# ./1
Alloca avg. cycles: 8954
Alloca elapsed time: 2.79792e-06
Malloc avg. cycles: 15152
Malloc avg. elapsed time: 4.73465e-06
```

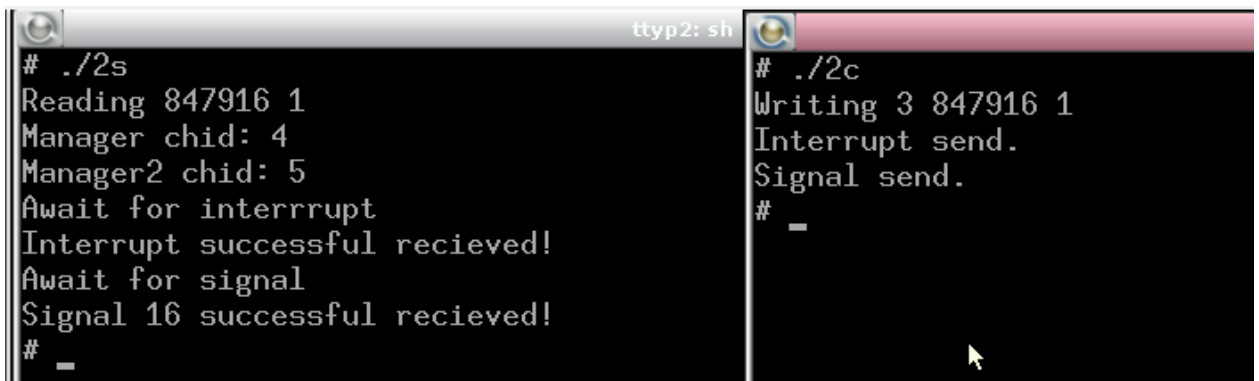
## Часть №2

Для проверки передачи сигнала и прерывания по сети были реализованы две программы – клиент (передатчик) и сервер (приемник). Поскольку простым способом сигналы и прерывания по сети передать нельзя, их необходимо было обернуть в специально событие – `sigevent` и использовать специальную функцию для передачи событий – `MsgDeliverEvent(int rcvid, const struct sigevent* ev)`, которая направляет событие по каналу от сервера к клиенту, где `rcvid` – идентификатор ранее принятого сообщения, `ev` - указатель на структуру с описанием события. Так же, чтобы сигнал и прерывание были доставлены, необходимо было инициализировать их с помощью методов `SIGEV_INTR_INIT(&ev)` и `SIGEV_SIGNAL_INIT(&ev, signo)`, где `signo` – тип сигнала, в моем случае я использовал из лекций пользовательский тип “SIGUSR1”:

```
SIGUSR1 = 16
SIGUSR2 = 17
```

сигналы, определяемые пользователем

. Чтобы принять сигнал я воспользовался методом `sigwait(const sigset_t *set, int *sig)` – ожидание сигнала, а уже для принятия прерывания методом от QNX – `InterruptWait(0, 0)` – ждет события типа `SIGEV_INTR`. Программы были скомпилированы и запущены поочередно, результат их работы:



```
# ./2s
Reading 847916 1
Manager chid: 4
Manager2 chid: 5
Await for interrrupt
Interrupt successful recieved!
Await for signal
Signal 16 successful recieved!
# -
```

```
# ./2c
Writing 3 847916 1
Interrupt send.
Signal send.
# -
```

Как видно на скриншоте передатчик записал информацию о подключении в /dev/shmem, сервер считал ее установив подключение, затем начал ожидать событие типа прерывание и успешно его получил от клиента, затем начал ждать событие типа сигнал и успешно его принял – номер сигнала 16, что совпадает с номером отправленного сигнала со скриншота выше.

### 3. Листинг

Часть№1 1.cpp:

```
#include <iostream>
#include <inttypes.h>
#include <malloc.h>
#include <sys/neutrino.h>
#include <sys/syspage.h>

#define CYCLES 100

int main()
{
    char *alloca_test, *malloc_test;
    uint64_t start, stop, cps;

    uint64_t alloca_time = 0;
    for(int i = 0; i < CYCLES; i++)
    {
        start = ClockCycles();
        alloca_test = (char *)alloca(100);
        stop = ClockCycles();
        alloca_time += stop - start;
    }
    std::cout << "Alloca avg. cycles: " << alloca_time / CYCLES <<
std::endl;
    cps = SYSPAGE_ENTRY(qtime)->cycles_per_sec;
    std::cout << "Alloca elapsed time: " <<
        (double)(alloca_time / CYCLES) / cps << std::endl;

    uint64_t malloc_time = 0;
    for(int i = 0; i < CYCLES; i++)
    {
        start = ClockCycles();
        malloc_test = (char *)malloc(100 * sizeof(char));
        stop = ClockCycles();
        malloc_time += stop - start;
        free(malloc_test);
    }
}
```

```

        std::cout << "Malloc avg. cycles: " << malloc_time / CYCLES <<
std::endl;
        cps = SYSPAGE_ENTRY(qtime)->cycles_per_sec;
        std::cout << "Malloc avg. elapsed time: " <<
            (double)(malloc_time / CYCLES) / cps << std::endl;

        return EXIT_SUCCESS;
    }

```

### Сервер (приемник) t2server.cpp:

```

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/neutrino.h>
#include <sys/netmgr.h>

int main()
{
    sigset_t set;
    int manager, manager2;
    int manager_chid = ChannelCreate(0);
    int manager2_chid;
    int pid = getpid(), pid2;
    int fd = open("/dev/shmem/NAME", O_RDWR);

    read(fd, &pid2, 4);
    read(fd, &manager2_chid, 4);
    printf("Reading %d %d\n", pid2, manager2_chid);

    manager = ConnectAttach(0, 0, manager_chid, 0, 0);
    printf("Manager chid: %d\n", manager);
    manager2 = ConnectAttach(ND_LOCAL_NODE, pid2, manager2_chid, 0, 0);
    printf("Manager2 chid: %d\n", manager2);

    printf("Await for interrrupt\n");
    MsgSend(manager2, 0, 0, 0, 0);
    if(InterruptWait(0, 0) != -1)
        printf("Interrupt successful recieved!\n");

    printf("Await for signal\n");
    sigemptyset(&set);
    sigaddset(&set, SIGUSR1);
    MsgSend(manager2, 0, 0, 0, 0);
    int sig = 0;
    if(sigwait(&set, &sig) == 0)
        printf("Signal %d successful recieved!\n", sig);
    return 0;
}

```

### Клиент (передатчик) t2client.cpp:

```

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/neutrino.h>
#include <sys/netmgr.h>

int main()
{
    struct sigevent event;
    int manager_chid = ChannelCreate(0);
    int manager = ConnectAttach(ND_LOCAL_NODE, 0, manager_chid, 0, 0);
    int pid = getpid(), pid2;
    int fd = open("/dev/shmem/NAME", O_CREAT+O_RDWR, 8000);

```

```

printf("Writing %d %d %d\n", manager, pid, manager_chid);
write(fd, &pid, 4);
write(fd, &manager_chid, 4);

int rcvid = MsgReceive(manager_chid, 0, 0, 0);
MsgReply(rcvid, 1, 0, 0);
sleep(1);
SIGEV_INTR_INIT(&event);
if(MsgDeliverEvent(rcvid, &event) != -1)
    printf("Interrupt send.\n");

rcvid = MsgReceive(manager_chid, 0, 0, 0);
MsgReply(rcvid, 1, 0, 0);
sleep(1);
SIGEV_SIGNAL_INIT(&event, SIGUSR1);
if(MsgDeliverEvent(rcvid, &event) != -1)
    printf("Signal send.\n");
return 0;
}

```