

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего
образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра комплексной информационной безопасности электронно-вычислительных
систем
(КИБЭВС)

Процессы

Отчёт по лабораторной работе №4
по дисциплине «Системное программирование»

Выполнил: Студент 717-1 гр.

Калжан К.Ж

«__»_____ 2021 г.

Проверил: Преподаватель

кафедры КИБЭВС

Полюга В.А

«__»_____ 2021 г.

1 Введение

Цель работы познакомиться с основными функциями WinAPI и POSIX API для работы с процессами, особенностями процессов в операционных системах Windows и Unix.

Задания:

1. Изучить краткие теоретические сведения, материалы лекций по теме практического занятия и приведенные примеры программ в методическом указании.

2. Используя Docker и соответствующий образ подготовить среду для разработки.

3. Реализовать программы, соответствующие вашему варианту задания на языке C++ для Linux (ваш вариант образа), в которых используются системные вызовы `fork()`, `exec()`, `wait()`, `exit()`, `kill()` и др. и продемонстрировать их работу.

4. Для вашего варианта языка программирования изучить встроенные высокоуровневые возможности языка программирования для работы с процессами и реализовать программы, соответствующие вашему варианту задания.

5. Научиться использовать команды `top`, `ps`, `kill`, `nice`, `export`, `set` (и другие, связанные с процессами), и изучить их параметры.

6. Написать отчет и защитить у преподавателя.

Вариант:

1) Образ докер - Ubuntu;

2) Язык программирования - Golang.

Индивидуальное задание: вариант 12

Описание задания: Написать две программы. Первая – вычисляет частоты встречаемости в тексте биграмм слов. Например, для предыдущего предложения биграммы слов это пары «первая вычисляет», «вычисляет частоты», «частоты встречаемости», «встречаемости в» и т.д. Вторая – принимает на вход текст, делит его на отдельные фрагменты и вычисляет частоты встречаемости в тексте биграмм слов, путем вызова первой программы для отдельных фрагментов текста.

2 Ход работы

2.1 Изучение теоретического материала

В ходе изучения теоретических материалов ознакомился с работой процессами на C++ в пункте 2.3 продемонстрирую работу с процессами на простых примерах, а после выполняем индивидуальную программу на высоком языке программирования.

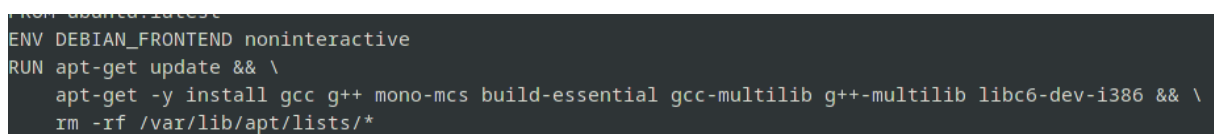
2.2 Подготовка образа

Подготовим образ операционной системы Linux для Docker для запуска программ:

- 1) Напишем Dockerfile;
- 2) Напишем программы по заданию.

Распишем составляющий Dockerfile:

- 1) Строиться образ будет на Ubuntu;
- 2) Пакеты которые установим для работы с C++(рисунок 2.1)



```
FROM ubuntu:14.04
ENV DEBIAN_FRONTEND noninteractive
RUN apt-get update && \
    apt-get -y install gcc g++ mono-mcs build-essential gcc-multilib g++-multilib libc6-dev-i386 && \
    rm -rf /var/lib/apt/lists/*
```

Рисунок 2.1 - установочные пакеты

2.3 Процессы на C++

Выполним простые программы которые показывают работу с процессами на C++ для понимания того что будет выполняться на других языках программирования

Для работы с процессами мы должны породить его, а после работать с ним. Чтобы породить процесс мы выполняем команду `fork` и присваиваем ее к переменной с типом `pid_t`. Когда создали процесс мы задаем ожидание `wait`, для завершения процесса и можно выполнить `exit`(рисунок 2.2).

```

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main()
{
    pid_t pid;
    int rv;
    switch(pid=fork()) {
    case -1:
        perror("fork"); /* произошла ошибка */
        exit(1); /*выход из родительского процесса*/
    case 0:
        printf(" CHILD: Это процесс-потомок!\n");
        printf(" CHILD: Мой PID -- %d\n", getpid());
        printf(" CHILD: PID моего родителя -- %d\n", getppid());
        printf(" CHILD: Введите мой код возврата(как можно меньше:");
        //scanf("%d");
        printf(" CHILD: Выход!\n");
        exit(rv);
    default:
        printf("PARENT: Это процесс-родитель!\n");
        printf("PARENT: Мой PID -- %d\n", getpid());
        printf("PARENT: PID моего потомка %d\n", pid);
        printf("PARENT: Я жду, пока потомок не вызовет exit()...\n");
        //wait();
        printf("PARENT: Код возврата потомка:%d\n", WEXITSTATUS(rv));
        printf("PARENT: Выход!\n");
    }
}

```

Рисунок 2.2 - Порождения процесса

Бывает задача при которой нужно передать поток в другой процесс. С этим помогает такая функция `exec`(рисунок 2.3). Кроме того возникает потребность изменить приоритет процесса.

```

// листинг 2ой программы
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(int argc, char *argv[], char *envp[])
{
    pid_t num;
    num = fork(); // порождаем новый процесс.
    if(num == 0)
    {
        execl("./test2", NULL, NULL);
        /* если дочерний процесс, то заменяем контекст доч
теперь запустилась 2-я программа */
    }
    else
    if(num > 0)
    {
        printf("Parent process\n\n");
    }
    return 0;
}

```

Рисунок 2.3 - Работа с `exec`

В ходе всего ознакомились с процессами теперь выполним в пункте 2.4 процессы для `golang`.

2.4 Процессы на Golang

В Go независимо запущенная задача называется горутинной. В данном уроке мы научимся запускать несколько горутин сразу и связывать их между собой через каналы. Горутинны похожи на корутинны, процессы или потоки в других языках, хотя у них есть много своих особенностей. Их создание рационально, оно значительно упрощает процесс управления многими конкурентными операциями.

Реализуем программу которая выполняет индивидуальное задание(Рисунок 2.4) без процессов.

```
package main

import (
    "fmt"
    "math"
    "strings"
    "unicode"
)
// SplitOnNonLetters splits a string on non-letter runes
func SplitOnNonLetters(s string) []string {
    notALetter := func(char rune) bool { return !unicode.IsLetter(char) }
    return strings.FieldsFunc(s, notALetter)
}

var str = "Мир - парк поезд. А куда ты! Мир парк"

func main() {
    str = strings.ToLower(str)
    parts := SplitOnNonLetters(str)
    fmt.Printf("%+v\n", parts)

    fmt.Println(ngrams(parts, 2))
}

func ngrams(words []string, size int) (count map[string]uint32) {
    count = make(map[string]uint32, 0)
    offset := int(math.Floor(float64(size / 2)))

    max := len(words)
    for i := range words {
        if i < offset || i+size-offset > max {
            continue
        }
        gram := strings.Join(words[i-offset:i+size-offset], " ")
        count[gram]++
    }
}
```

Рисунок 2.4 - Программа на Golang

Добавим горутинны в программу(рисунок 2.5)

```

func SplitOnNonLetters(s string) []string {
    notALetter := func(char rune) bool { return !unicode.IsLetter(char) }
    return strings.FieldsFunc(s, notALetter)
}

var str = "Мир - парк поезд. А куда ты! Мир парк"
//var str1 = "Мир - парк1 поезд. А куда ты! Мир парк"
func main() {
    fmt.Println(str)
    //text := map[string]uint32
    test := make(chan map[string]uint32)
    words := regexp.MustCompile("[.?!]{1} ").Split(str, -1)
    for _, word := range words {
        str = strings.ToLower(word)
        parts := SplitOnNonLetters(str)
        go ngrams(parts, 2, test)
    }
    for i, _ := range words {
        fmt.Println(i)
        t := <- test
        fmt.Println(t)
    }
}

func ngrams(words []string, size int, test chan map[string]uint32) {
    count := make(map[string]uint32, 0)
    offset := int(math.Floor(float64(size / 2)))

    max := len(words)
    for i := range words {
        if i < offset || i+size-offset > max {
            continue
        }
        gram := strings.Join(words[i-offset:i+size-offset], " ")
        count[gram]++
    }
    test <- count
}

```

Рисунок 2.5 - Применения горутины

Результаты работы программ представлена на рисунке 2.6

```

~/СП/lab_4 >>> go run main.go
[мир парк поезд а куда ты мир парк]
map[a куда:1 куда ты:1 мир парк:2 парк поезд:1 поезд а:1 ты мир:1]
~/СП/lab_4 >>> |

```

Рисунок 2.6 - Результаты программы

2.5 Утилиты для работы с процессами

Поработаем с утилитами для работы с процессами:

1) Первая утилита для работы с процессами “PS”, которая служит для просмотра запущенных процессов(рисунок 2.7);

```

~/СП/lab_4 >>> ps
  PID TTY          TIME CMD
 2262 pts/0        00:00:04 zsh
 24600 pts/0        00:00:00 ps
~/СП/lab_4 >>> |

```

Рисунок 2.7 - Работа с утилитой ps

2) Вторая утилита которая позволяет просмотреть работу процессов но уже обширно top(рисунок 2.8);

```
top - 13:30:46 up 1:22, 1 user, load average: 0.95, 0.94, 0.78
Tasks: 233 total, 1 running, 232 sleeping, 0 stopped, 0 zombie
%Cpu(s): 1.2 us, 1.3 sy, 0.0 ni, 97.5 id, 0.0 wa, 0.0 hi, 0.1 si, 0.0 st
MiB Mem : 15439.5 total, 11303.9 free, 2563.9 used, 1571.8 buff/cache
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 12296.1 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1730	hanmask	20	0	5048548	335916	157788	S	8.0	2.1	6:04.04	gnome-shell
2906	hanmask	20	0	3100076	420572	180800	S	2.0	2.7	3:59.89	Web Content
3046	hanmask	20	0	2943648	320032	192468	S	1.7	2.0	3:50.72	Web Content
6480	hanmask	20	0	2884388	236144	157676	S	1.0	1.5	0:35.90	Web Content
921	root	20	0	1567068	45488	24152	S	0.7	0.3	0:19.59	containerd
2185	hanmask	20	0	342840	28572	16156	S	0.7	0.2	0:06.51	ibus-extension-
2474	hanmask	20	0	3664772	491404	232464	S	0.7	3.1	8:05.30	firefox
2731	hanmask	20	0	3384216	650008	241560	S	0.7	4.1	8:21.00	Web Content
3090	hanmask	20	0	2512892	163060	110996	S	0.7	1.0	0:24.66	Web Content
7441	hanmask	20	0	2543452	191648	138040	S	0.7	1.2	0:28.58	Web Content
28	root	20	0	0	0	0	S	0.3	0.0	0:16.69	ksoftirqd/2
529	root	-51	0	0	0	0	S	0.3	0.0	0:17.13	irq/78-rtw88_pc
1789	hanmask	20	0	1228492	123028	86288	S	0.3	0.8	1:15.81	Xwayland
1888	hanmask	20	0	314904	7932	7068	S	0.3	0.1	0:00.31	gvfs-afc-volume
2178	hanmask	20	0	310236	10840	6392	S	0.3	0.1	0:40.94	ibus-daemon
2257	hanmask	20	0	413492	61292	48076	S	0.3	0.4	0:06.86	gnome-terminal-
6530	hanmask	20	0	2835872	271640	181932	S	0.3	1.7	0:18.11	Web Content
23759	hanmask	20	0	10580	4180	3376	R	0.3	0.0	0:00.05	top
1	root	20	0	171196	10748	7980	S	0.0	0.1	0:00.81	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-kblockd
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
9	root	20	0	0	0	0	S	0.0	0.0	0:01.44	ksoftirqd/0
10	root	-2	0	0	0	0	S	0.0	0.0	0:00.00	rcuc/0
11	root	-2	0	0	0	0	I	0.0	0.0	0:11.13	rcu_preempt
12	root	-2	0	0	0	0	S	0.0	0.0	0:00.00	rcub/0
13	root	rt	0	0	0	0	S	0.0	0.0	0:00.02	migration/0

Рисунок 2.8 - Работа с утилитой top

3) Ещё одна утилита которая позволяет не с процессами на прямую а больше с пространством глобальных переменных утилита export(рисунок 2.9);

```
~/СП/lab_4 >>> export DATE_BUILD="$(date +%F)"
~/СП/lab_4 >>> echo $DATE_BUILD
2021-03-14
~/СП/lab_4 >>> |
```

Рисунок 2.9 - Работа с export

4) Ещё одна утилита которая также помогает работу с пространством переменных утилита set(рисунок 2.10). Set нужно для просмотра побольше части.

```
2021-03-14
~/СП/lab_4 >>> set
'!' = 0
'#' = 0
'$' = 2262
'*' = ( )
-=0569BJPXZims
0=zsh
'? '=0
@=( )
ARGC=0
PHEEP 11
```

Рисунок 2.10 - Работа с командой set

3 Заключение

В ходе лабораторной работы познакомились с основными функциями WinAPI и POSIX API для работы с процессами, особенностями процессов в операционных системах Windows и Unix.

Выполнили следующие пункты:

1. Изучил краткие теоретические сведения, материалы лекций по теме практического занятия и приведенные примеры программ в методическом указании.
2. Создал докер образ соответствующий для разработки.
3. Реализовал программы, соответствующему варианту(12 вариант) задания на языке C++ для Linux (ваш вариант образа), в которых используются системные вызовы `fork()`, `exec()`, `wait()`, `exit()`, `kill()` и др. и продемонстрировал их работу.
4. Для своего варианта языка программирования изучил встроенные высокоуровневые возможности языка программирования для работы с процессами и реализовал программу, соответствующему варианту задания.
5. Научился использовать команды `top`, `ps`, `kill`, `nice`, `export`, `set`(и другие, связанные с процессами), и изучил их параметры.

Список источников

1) Github [Электронный ресурс] - Режим доступа:
https://github.com/kasymhan/sp_lab_4 (дата обращения: 16.03.2021).

Приложение А

```
#include <stdio.h>  
#include <stdlib.h>
```

```

int main(){
    // переменные
    int m,n,t,i,j,k;
    m = 9;
    n = 10;
    int range = 10;
    int A[m][n];
    int B[n][1];
    int C[m][1];
    // Заполнение переменных матрицы A
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++){
            A[i][j]= rand() % range+1;
        }
    }
    // Заполнение переменных матрицы B
    for(int i=0;i<n;i++){
        B[i][0]=rand() % range+1;
    }

    printf("Матрица A\n");
    for (int i=0; i<m; i++){
        for(int j=0; j<n; j++){

            printf("%d\t", A[i][j]);

        }
        printf("\n");
    }
    printf("Матрица B\n");
    for (int i=0;i<n;i++){
        printf("%d\n",B[i][0]);
    }
    int *ptr_array_a = &A[0][0];
    int *ptr_array_b = &B[0][0];
    int *ptr_array_c = &C[0][0];
    int *end_array_a = &A[m-1][n-1];

```

```

int *end_array_b = &B[n-1][1];
int *end_array_c = &C[m-1][1];
// Перемножение матриц
i=0;
j=0;
k=0;
asm(
"mov %[PTRA], %%ebx\n"
//Суммирование столбцов по строке
"loop:\n"
"mov $4, %%eax\n"
"mov %[i], %%edx\n"
"mov %%edx, %%ecx\n"
"mul %%ecx\n"
"mov %[PTRB], %%edx\n"
"add %%eax, %%edx\n"
"mov (%%edx), %%eax\n"
"mov (%%ebx), %%ecx\n"
"mul %%ecx\n"
"add %%eax, %[k]\n"
"addl $1,%[i]\n"
"mov %[i], %%edx\n"
"mov %[n], %%ecx\n"
"add $4, %%ebx\n"
"cml $4, %%ecx\n"
"jne loop\n"
// Переход на следующую строку
"movl $0,%[i]\n"
"mov %[j], %%edx\n"
"movl $4, %%eax\n"
"mov %%edx, %%ecx\n"
"mul %%ecx\n"
"mov %[PTRC],%%edx\n"
"add %%eax, %%edx\n"
"mov %[k], %%ecx\n"
"mov %%ecx, (%%edx)\n"
"movl $0, %[k]\n"
"addl $1, %[j]\n"

```

```

"mov %[j], %%edx\n"
"mov %[m], %%ecx\n"
"cmpl %%edx, %%ecx\n"
"jne loop\n"
:
:
[n]"m"(n),
[i]"m"(i),
[j]"m"(j),
[PTRA]"m"(ptr_array_a),
[PTRB]"m"(ptr_array_b),
[PTRC]"m"(ptr_array_c),
[ENDA]"m"(end_array_a),
[ENDB]"m"(end_array_b),
[ENDC]"m"(end_array_c),
[m]"m"(m),
[k]"m"(k)
: "%edx", "%ebx", "%eax", "%ecx"
);
// Получившая матрица
printf("Новая матрица\n");
for(int i=0;i<m;i++){
printf("%d\n",C[i][0]);
}
return 0;
}

```