Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ  
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

[Кафедра программной инженерии](https://www.belstu.by/fakultety/fit/vm)

Специальность 1-40 01 01 Программное обеспечение информационных технологий

**Отчёт по лабораторной работе №5**

Дисциплина: Операционные системы

Тема: Диспетчеризация

Выполнила:

студентка 3 курса 5 группы

Вовна Ярослава Руслановна

Минск 2024

**Задание 01**

int main()

{

HANDLE processHandle = GetCurrentProcess();

HANDLE threadHandle = GetCurrentThread();

DWORD pid = GetCurrentProcessId();

DWORD tid = GetCurrentThreadId();

cout << "Main PID: " << pid << endl;

cout << "Main TID: " << tid << endl;

DWORD processPriority = GetPriorityClass(processHandle);

switch (processPriority)

{

case IDLE\_PRIORITY\_CLASS: cout << "Process priority: IDLE\n"; break;

case NORMAL\_PRIORITY\_CLASS: cout << "Process priority: NORMAL\n"; break;

case HIGH\_PRIORITY\_CLASS: cout << "Process priority: HIGH\n"; break;

case REALTIME\_PRIORITY\_CLASS: cout << "Process priority: REALTIME\n"; break;

case BELOW\_NORMAL\_PRIORITY\_CLASS: cout << "Process priority: BELOW\_NORMAL\n"; break;

case ABOVE\_NORMAL\_PRIORITY\_CLASS: cout << "Process priority: ABOVE\_NORMAL\n"; break;

default: cout << "Process priority: ???\n"; break;

}

DWORD threadPriority = GetThreadPriority(threadHandle);

switch (threadPriority)

{

case THREAD\_PRIORITY\_LOWEST: cout << "Thread priority: LOWEST\n"; break;

case THREAD\_PRIORITY\_BELOW\_NORMAL: cout << "Thread priority: TBELOW\_NORMAL\n"; break;

case THREAD\_PRIORITY\_NORMAL: cout << "Thread priority: NORMAL\n"; break;

case THREAD\_PRIORITY\_ABOVE\_NORMAL:cout << "Thread priority: ABOVE\_NORMAL\n"; break;

case THREAD\_PRIORITY\_HIGHEST: cout << "Thread priority: HIGHEST\n"; break;

case THREAD\_PRIORITY\_IDLE: cout << "Thread priority: IDLE\n"; break;

default:cout << "Thread priority: ???\n"; break;

}

DWORD\_PTR pa = NULL, sa = NULL;

char buf[10];

if (!GetProcessAffinityMask(processHandle, &pa, &sa))

throw "Error";

\_itoa\_s(pa, buf, 2);

cout << "Process affinity mask: " << buf;

cout << " (" << showbase << hex << pa << ")" << endl;

\_itoa\_s(sa, buf, 2);

SYSTEM\_INFO sys\_info;

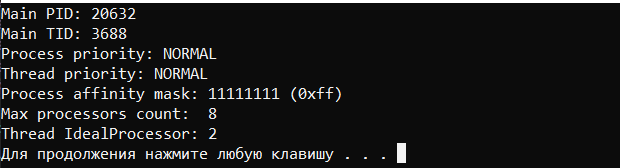
GetSystemInfo(&sys\_info);

cout << "Max processors count: " << dec << sys\_info.dwNumberOfProcessors << endl;

cout << "Thread IdealProcessor: " << dec << SetThreadIdealProcessor(threadHandle, MAXIMUM\_PROCESSORS) << endl;

system("pause");

}



**Задание 02**

2-3)

int main()

{

HANDLE processHandle = GetCurrentProcess();

HANDLE threadHandle = GetCurrentThread();

DWORD pid = GetCurrentProcessId();

DWORD tid = GetCurrentThreadId();

DWORD\_PTR icpu = -1;

for (int i = 0; i < 1000000; i++)

{

if (i % 1000 == 0 && i != 0)

{

Sleep(200);

cout << "Iteration number: " << dec << i << endl;

cout << "Main PID: " << dec << pid << endl;

cout << "Main TID: " << dec << tid << endl;

DWORD processPriority = GetPriorityClass(processHandle);

switch (processPriority)

{

case IDLE\_PRIORITY\_CLASS: cout << "Process priority: IDLE\n"; break;

case NORMAL\_PRIORITY\_CLASS: cout << "Process priority: NORMAL\n"; break;

case HIGH\_PRIORITY\_CLASS: cout << "Process priority: HIGH\n"; break;

case REALTIME\_PRIORITY\_CLASS: cout << "Process priority: REALTIME\n"; break;

case BELOW\_NORMAL\_PRIORITY\_CLASS: cout << "Process priority: BELOW\_NORMAL\n"; break;

case ABOVE\_NORMAL\_PRIORITY\_CLASS: cout << "Process priority: ABOVE\_NORMAL\n"; break;

default: cout << "Process priority: ???\n"; break;

}

DWORD threadPriority = GetThreadPriority(threadHandle);

switch (threadPriority)

{

case THREAD\_PRIORITY\_LOWEST: cout << "Thread priority: LOWEST\n"; break;

case THREAD\_PRIORITY\_BELOW\_NORMAL: cout << "Thread priority: TBELOW\_NORMAL\n"; break;

case THREAD\_PRIORITY\_NORMAL: cout << "Thread priority: NORMAL\n"; break;

case THREAD\_PRIORITY\_ABOVE\_NORMAL:cout << "Thread priority: ABOVE\_NORMAL\n"; break;

case THREAD\_PRIORITY\_HIGHEST: cout << "Thread priority: HIGHEST\n"; break;

case THREAD\_PRIORITY\_IDLE: cout << "Thread priority: IDLE\n"; break;

default:cout << "Thread priority: ???\n"; break;

}

icpu = SetThreadIdealProcessor(threadHandle, MAXIMUM\_PROCESSORS);

cout << "Thread IdealProcessor: " << dec << icpu << endl;

}

}

system("pause");

}



DWORD getProcessPriority(int i)

{

switch (i)

{

case 1: return IDLE\_PRIORITY\_CLASS;

case 2: return BELOW\_NORMAL\_PRIORITY\_CLASS;

case 3: return NORMAL\_PRIORITY\_CLASS;

case 4: return ABOVE\_NORMAL\_PRIORITY\_CLASS;

case 5: return HIGH\_PRIORITY\_CLASS;

case 6: return REALTIME\_PRIORITY\_CLASS;

default: throw "???";

}

}

int main(int argc, char\* argv[])

{

try

{

if (argc == 4)

{

HANDLE hp = GetCurrentProcess();

DWORD\_PTR mask = atoi(argv[1]);

DWORD priority1 = atoi(argv[2]);

DWORD priority2 = atoi(argv[3]);

if (!SetProcessAffinityMask(hp, mask)) throw "Error";

cout << "Current affinity mask: " << showbase << hex << mask << endl;

cout << "Process priority 1: " << getProcessPriority(priority1) << endl;

cout << "Process priority 2: " << getProcessPriority(priority2) << endl;

LPCWSTR path = L"D:\\лабы\\oc\\lab5\\lab5\\Debug\\os05\_02x.exe";

STARTUPINFO si1, si2;

PROCESS\_INFORMATION pi1, pi2;

ZeroMemory(&si1, sizeof(STARTUPINFO));

ZeroMemory(&si2, sizeof(STARTUPINFO));

si1.cb = sizeof(STARTUPINFO);

si2.cb = sizeof(STARTUPINFO);

if (CreateProcess(path, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | getProcessPriority(priority1), NULL, NULL, &si1, &pi1))

cout << "os5\_02-1 was created\n";

else cout << "error with creating os05\_02-1\n";

if (CreateProcess(path, NULL, NULL, NULL, FALSE, CREATE\_NEW\_CONSOLE | getProcessPriority(priority2), NULL, NULL, &si2, &pi2))

cout << "os5\_02-2 was created\n";

else cout << "error with creating os05\_02-2\n";

WaitForSingleObject(pi1.hProcess, INFINITE);

WaitForSingleObject(pi2.hProcess, INFINITE);

CloseHandle(pi1.hProcess);

CloseHandle(pi2.hProcess);

}

else

cout << "Need for 4 args" << endl;

}

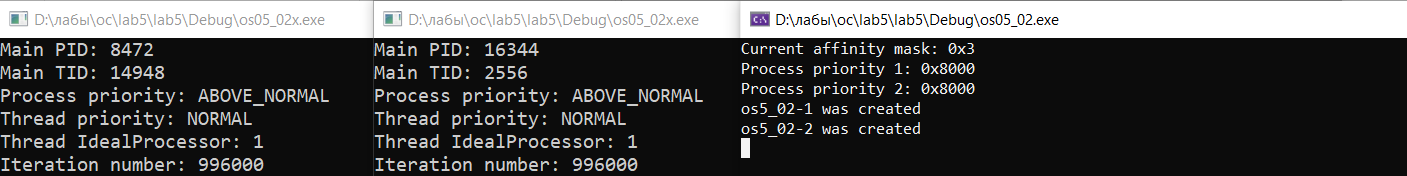
catch (string err)

{

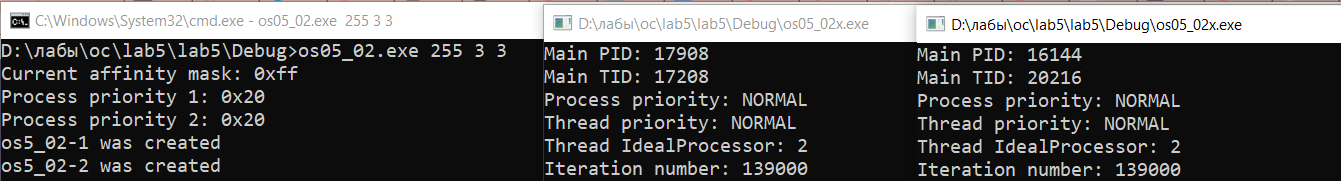
cout << err << endl;

}

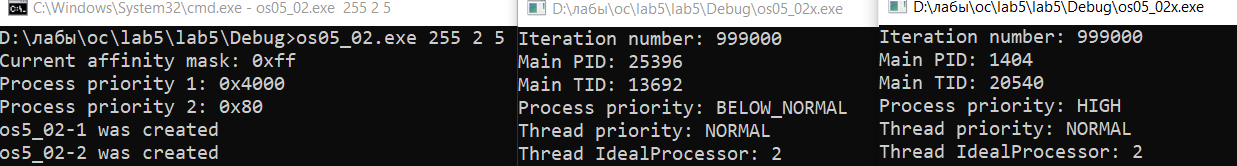
}



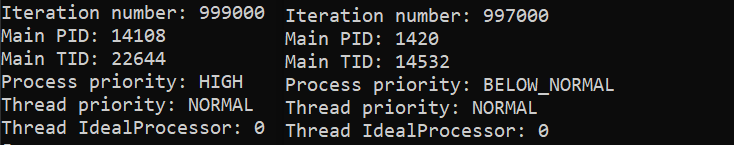
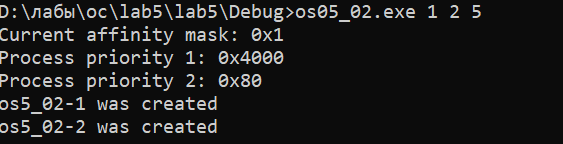
6) 255 3 3



7) 255 2 5



8) 1 2 5



9) В случаях 6 и 7 процессы будут работать одинаково быстро, потому как процессам доступны все допустимые процессоры и они не загружены. А в 8 появляется конкуренция за ресурсы, поэтому появляется разница в производительности.

**Задание 03**

10-12)

DWORD getProcessPriority(int i)

{

switch (i)

{

case 1: return IDLE\_PRIORITY\_CLASS;

case 2: return BELOW\_NORMAL\_PRIORITY\_CLASS;

case 3: return NORMAL\_PRIORITY\_CLASS;

case 4: return ABOVE\_NORMAL\_PRIORITY\_CLASS;

case 5: return HIGH\_PRIORITY\_CLASS;

case 6: return REALTIME\_PRIORITY\_CLASS;

default: throw "???";

}

}

DWORD getThreadPriority(int i)

{

switch (i)

{

case 1: return THREAD\_PRIORITY\_LOWEST;

case 2: return THREAD\_PRIORITY\_BELOW\_NORMAL;

case 3: return THREAD\_PRIORITY\_NORMAL;

case 4: return THREAD\_PRIORITY\_ABOVE\_NORMAL;

case 5: return THREAD\_PRIORITY\_HIGHEST;

case 6: return THREAD\_PRIORITY\_IDLE;

default: throw "???";

}

}

void TA()

{

DWORD pid = GetCurrentProcessId();

DWORD tid = GetCurrentThreadId();

HANDLE hp = GetCurrentProcess();

HANDLE ht = GetCurrentThread();

for (int i = 0; i < 1000000; i++)

{

if (i % 1000 == 0 && i != 0)

{

Sleep(200);

cout << "Iteration number: " << dec << i << endl;

cout << "Main PID: " << dec << pid << endl;

cout << "Main TID: " << dec << tid << endl;

DWORD processPriority = GetPriorityClass(hp);

switch (processPriority)

{

case IDLE\_PRIORITY\_CLASS: cout << "Process priority: IDLE\n"; break;

case BELOW\_NORMAL\_PRIORITY\_CLASS: cout << "Process priority: BELOW\_NORMA\n"; break;

case NORMAL\_PRIORITY\_CLASS: cout << "Process priority: NORMAL\n"; break;

case ABOVE\_NORMAL\_PRIORITY\_CLASS: cout << "Process priority: ABOVE\_NORMAL\n"; break;

case HIGH\_PRIORITY\_CLASS: cout << "Process priority: HIGH\n"; break;

case REALTIME\_PRIORITY\_CLASS: cout << "Process priority: REALTIME\n"; break;

default: cout << "Process priority: ???\n"; break;

}

DWORD threadPriority = GetThreadPriority(ht);

switch (threadPriority)

{

case THREAD\_PRIORITY\_LOWEST: cout << "Thread priority: LOWEST\n"; break;

case THREAD\_PRIORITY\_BELOW\_NORMAL: cout << "Thread priority: TBELOW\_NORMAL\n"; break;

case THREAD\_PRIORITY\_NORMAL: cout << "Thread priority: NORMAL\n"; break;

case THREAD\_PRIORITY\_ABOVE\_NORMAL:cout << "Thread priority: ABOVE\_NORMAL\n"; break;

case THREAD\_PRIORITY\_HIGHEST: cout << "Thread priority: HIGHEST\n"; break;

case THREAD\_PRIORITY\_IDLE: cout << "Thread priority: IDLE\n"; break;

default:cout << "Thread priority: ???\n"; break;

}

DWORD icpu = SetThreadIdealProcessor(ht, MAXIMUM\_PROCESSORS);

cout << "Thread IdealProcessor: " << dec << icpu << endl;

}

}

cout << "End of routine" << endl;

}

int main(int argc, char\* argv[])

{

try

{

if (argc == 5)

{

HANDLE hp = GetCurrentProcess();

DWORD mask = atoi(argv[1]);

DWORD processPriority = atoi(argv[2]);

DWORD priority1 = atoi(argv[3]);

DWORD priority2 = atoi(argv[4]);

if (!SetProcessAffinityMask(hp, mask))

throw "SetProcessAffinityMask error";

cout << "Current affinity mask: " << showbase << hex << mask << endl;

cout << "Process priority: " << getProcessPriority(processPriority) << endl;

cout << "Thread priority 1: " << getThreadPriority(priority1) << endl;

cout << "Thread priority 2: " << getThreadPriority(priority2) << endl;

DWORD childId\_T1, childId\_T2 = NULL;

HANDLE hChild1 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)TA, NULL, CREATE\_SUSPENDED, &childId\_T1);

HANDLE hChild2 = CreateThread(NULL, 0, (LPTHREAD\_START\_ROUTINE)TA, NULL, CREATE\_SUSPENDED, &childId\_T2);

SetThreadPriority(hChild1, getThreadPriority(priority1));

SetThreadPriority(hChild2, getThreadPriority(priority2));

ResumeThread(hChild1);

ResumeThread(hChild2);

WaitForSingleObject(hChild1, INFINITE);

WaitForSingleObject(hChild2, INFINITE);

CloseHandle(hChild1);

CloseHandle(hChild2);

}

else

cout << "Need for 4 args" << endl;

}

catch (string err)

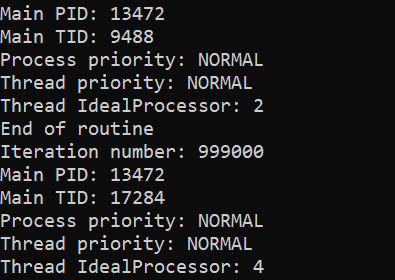
{

cout << err << endl;

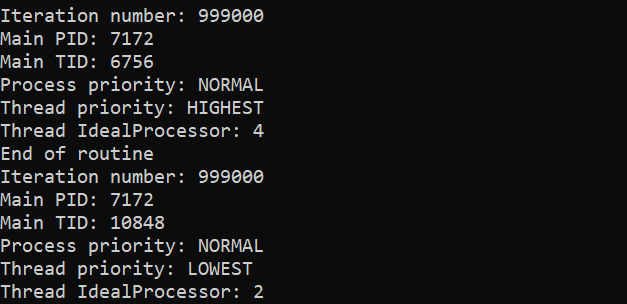
}

}

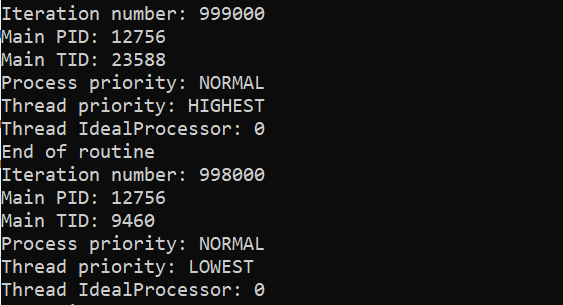
13) 255 3 3 3



14) 255 3 1 5



15) 1 3 1 5



16) Объяснение аналогично пункту 9

**Задание 04**

17)

int main()

{

pid\_t pid = getpid();

pid\_t tid = gettid();

int nicity = nice(0);

cpu\_set\_t set;

CPU\_ZERO(&set);

printf("PID = %d\nTID = %d\nNice = %d\n", pid, tid, nicity);

if (sched\_getaffinity(0, sizeof(cpu\_set\_t), &set) == 0)

{

for (int i = 0; i < CPU\_SETSIZE; ++i)

{

if (CPU\_ISSET(i, &set))

printf("CPU = %i is set\n", i);

}

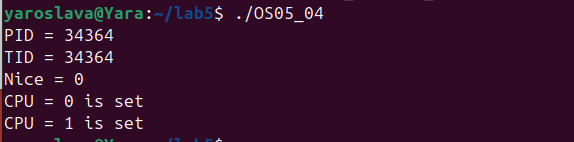
}

else

printf("shed\_getaffinity returned -1.");

exit(0);

}



**Задание 05**

18)

int main()

{

pid\_t pid = getpid();

nice(0);

for(int i = 0; i < 1000000000; i++) {

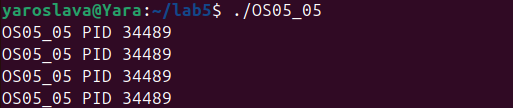
printf("OS05\_05 PID %d\n", pid);

sleep(1);

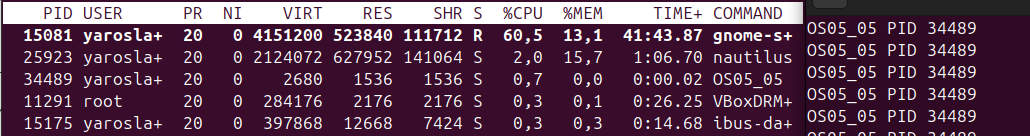
}

exit(0);

}



20)



21) sudo renice -20 –p 34489



22) sudo renice 19 –p 34489



**Задание 06**

23)

int main()

{

pid\_t pid = getpid();

pid\_t tid = gettid();

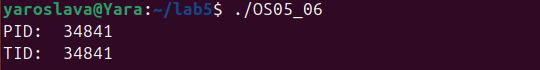
for (int i = 0; i < 1000000; ++i)

{

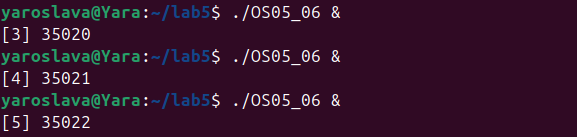
sleep(1);

}

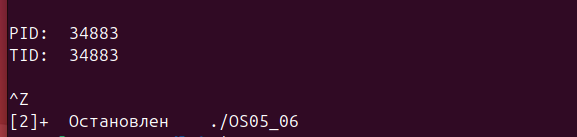
}



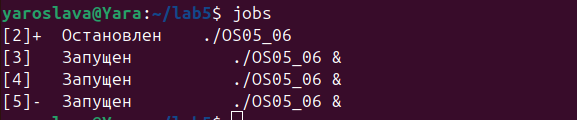
24) запуск в фоновом режиме



Ctrl + Z



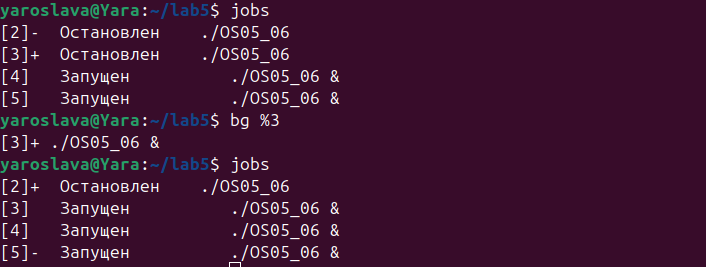
jobs



Kill -9



Bg



Fg



**Задание 07**

25)

void thread1()

{

pid\_t pid = getpid();

int cnice = nice(0);

printf("Nice: %d\n", cnice);

for (int i = 0; i < 1000000; ++i)

{

if (i == 1)

{

cnice = nice(10);

}

printf("Nice: %d\n", cnice);

printf("Child PID: %d\n", pid);

sleep(1);

}

}

void thread2()

{

pid\_t pid = getpid();

int cnice = nice(0);

for (int i = 0; i < 1000000; i++)

{

printf("Main nice: %d\n", cnice);

printf("Main PID: %d\n", pid);

sleep(1);

}

}

int main()

{

pid\_t pid;

switch (pid = fork())

{

case -1:

perror("Fork() returned -1.");

exit(-1);

case 0:

thread1();

exit(0);

default:

thread2();

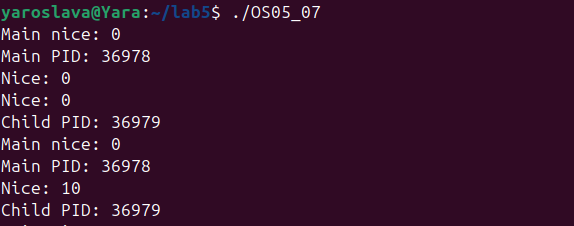
wait(NULL);

}

exit(0);

}

26)



watch -n 1 "ps -eo pid,ni,cmd | grep OS05\_07"

