2021270660 이지원 - 운영체제 실습 과제 4
Process Tree

Table of contents

System documentation

```
#define _PROCESS_H_
#define _PSTREE_H_
#define _CRT_SECURE_NO_WARNINGS


#include <iostream>

// to get num of running processes
#include <wtypes.h>
#include <Psapi.h>

#include <TlHelp32.h> // to use createToolhelp32Snapshot()
#include <vector>

#include <tchar.h> // to use TCHAR
#include <wtypes.h> // to use DWORD
```

필요한 라이브러리 불러오기 및 DEFINE 부분

System documentation

```
// Process 클래스 정의
class Process {
    TCHAR* name; // 프로세스의 이름
    DWORD pid, ppid; // 프로세스 id와 parent 프로세스 id 번호
    DWORD newPpid; // parent 프로세스가 죽을 경우 새로 parent process가 되는 root 프로세스
    std::vector<Process> children; // child 프로세스 목록

public:
    Process(TCHAR name[], DWORD pid, DWORD ppid) {
        int nameLength = _tcslen(name);
        this->name = new TCHAR[nameLength + 1];
        wcscpy(this->name, name); // 이름 복사
        this->pid = pid;
        this->ppid = ppid;
        newPpid = ppid;
    }
};
```

Process 클래스 선언 부분 → 프로세스 정보 출력에 필요한 정보 멤버화
포인터와 wcscpy를 통해 프로세스 명 담아두기

```
    /// get methods
    Process* getChild(int i) { return &children[i]; };
    int getSizeOfChildren() { return children.size(); };
    DWORD getPID() { return pid; };
    DWORD getPPID() { return ppid; };

    // set methods
    void setNewPPID(DWORD newPpid) { this->newPpid = newPpid; };
    void addChild(Process p) { children.push_back(p); };

    // child 프로세스 존재 여부
    bool hasChild() {
        if (children.size()) return true;
        return false;
    };

    // 프로세스 정보(이름, pid, root pid, ppid ) 출력
    void printProcInfo() {
        printf("+-%S(%d) (%d:%d)\n", name, pid, newPpid, ppid);
    };
};
```

퍼블릭 멤버 함수 정의부 (get/set, child 여부, 출력)

System documentation

PSTree 클래스 선언부 및 멤버 함수 정의

```
// PSTree 클래스 정의
class PSTree {
    Process* root; // root process ( init process )

public:
    // 생성자 함수
    // root process 초기화
    PSTree(TCHAR nameOfRoot[], DWORD pidOfRoot, DWORD ppidOfRoot) {
        root = new Process(nameOfRoot, pidOfRoot, ppidOfRoot);
    }

    // parent 프로세스를 찾는 함수
    // input : current process의 주소, ppid
    // output : 부모 프로세스의 주소
    //         부모 프로세스가 죽어 찾을 수 없을 경우, return null
    Process* findParentProcess(Process* currentProcess, DWORD ppid) {
        // 현재 프로세스가 parent process일 경우 return address of current process
        if (currentProcess->getPID() == ppid) {
            return currentProcess;
        }

        // child가 있는 경우
        if (currentProcess->hasChild()) {
            Process* child = NULL;
            Process* parent = NULL;

            // current 프로세스의 child를 순회하며 parent process 찾기
            for (std::vector<Process>::size_type iOfChild = 0; iOfChild < currentProcess->getSizeOfChildren(); iOfChild++) {
                child = currentProcess->getChild(iOfChild);
                if (parent = findParentProcess(child, ppid)) // 찾으면 return
                    return parent;
            }

            // 찾을 수 없을 경우 return null
            return NULL;

        } else {
            return NULL;
        }
    }
    ;
```

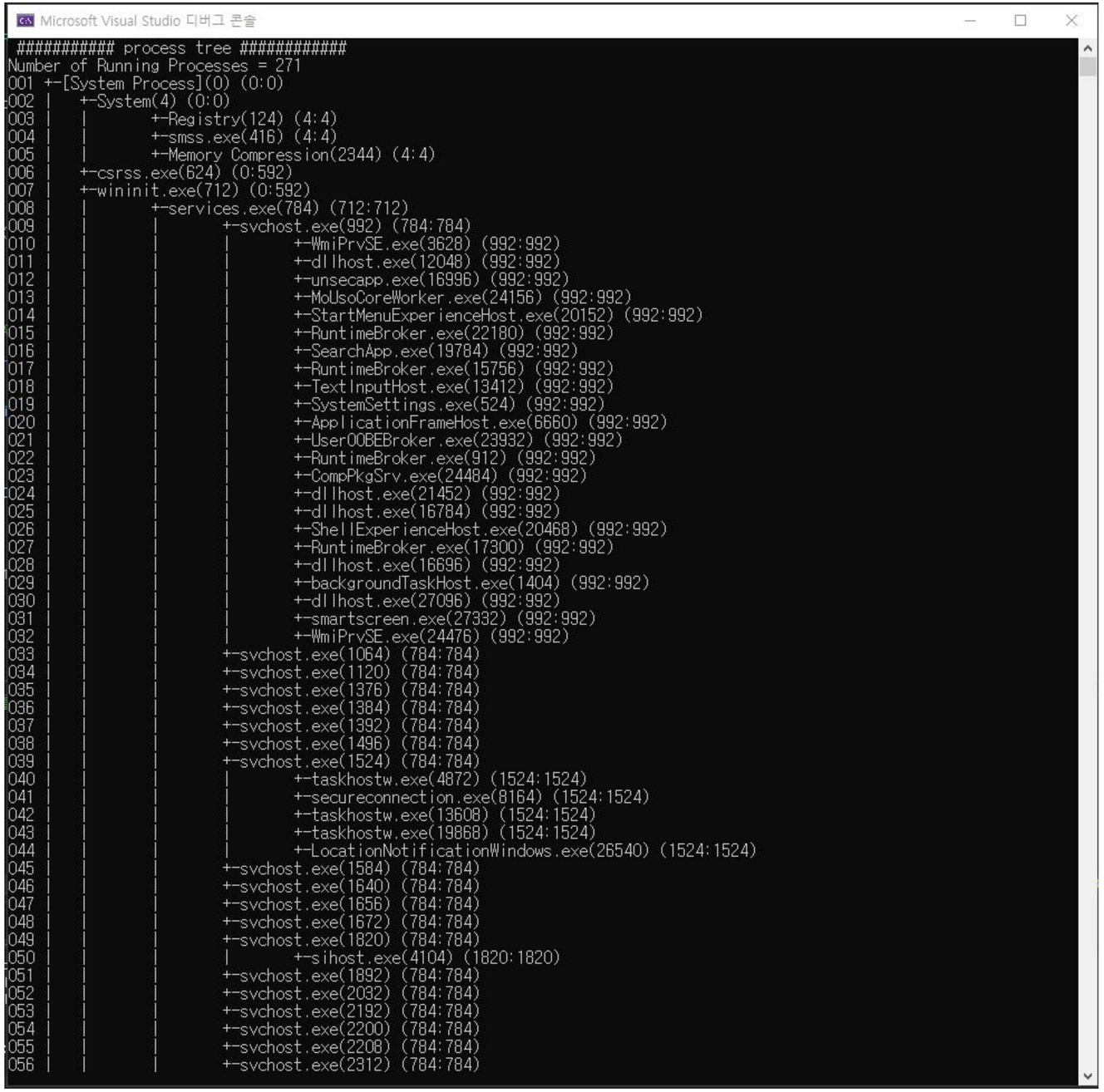
```
    // 프로세스 트리에 프로세스 추가하는 함수
    // input : new process의 정보
    void addProc(TCHAR name[], DWORD pid, DWORD ppid) {
        Process newProc(name, pid, ppid);

        // new 프로세스의 parent 찾기
        Process* parent = findParentProcess(root, newProc.getPPID());

        // 찾을 수 없을 경우, 루트 프로세스로 설정
        if (parent == NULL) {
            newProc.setNewPPID(0);
            root->addChild(newProc);

        } else { // 찾은 경우 parent에 추가
            newProc.setNewPPID(parent->getPID());
            parent->addChild(newProc);
        }
    }
    ;

    // input : current 프로세스 주소, 출력을 위한 indentations
    // output : 프로세스 정보 출력
    void printProcess(Process* currentProc, int indent, int* iOfProc) {
        // child가 있을 경우 순회하며 출력
        Process* child;
        for (std::vector<Process>::size_type iOfChild = 0; iOfChild < currentProc->getSizeOfChildren(); iOfChild++) {
            printf("%03d ", (*iOfProc)++);
            for (int j = 0; j <= indent; j++) {
                printf("|\t");
            }
            child = currentProc->getChild(iOfChild);
            child->printProcInfo();

            // child의 child도 출력 ...
            if (child->hasChild()) {
                printProcess(child, indent + 1, iOfProc);
            }
        }
        indent++;
    };

    // process tree 출력
    void printPSTree() {
        int iOfProc = 1; // 출력되는 프로세스들의 index

        // 루트 프로세스 정보 먼저 출력
        printf("%03d ", iOfProc++);
        root->printProcInfo();

        // 나머지 프로세스 출력
        printProcess(root, 0, &iOfProc);
    };
};
```

System documentation

실행되는 main 함수

```
int main() {

    // identifier 선언
    HANDLE hProcessSnap; // 프로세스를 snap
    PROCESSENTRY32 pe32; // 프로세스의 정보를 담는 구조체
    int numOfProc = 0; // 총 프로세스의 수

    // snap process
    hProcessSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (hProcessSnap == INVALID_HANDLE_VALUE) {
        puts("CreateToolhelp32Snapchot error");
        exit(EXIT_FAILURE);
    }

    // init 프로세스 정보 가져오기
    pe32.dwSize = sizeof(PROCESSENTRY32);
    if (!Process32First(hProcessSnap, &pe32)) {
        puts("Process32First error!");
        CloseHandle(hProcessSnap);
        exit(EXIT_FAILURE);
    }
    numOfProc++; // 프로세스 수 1 증가

    // 모든 프로세스 가져오기
    PSTree pstree(pe32.szExeFile, pe32.th32ProcessID, pe32.th32ParentProcessID); // init 프로세스 = root

    // 다음 프로세스들의 정보를 받아 트리 만들기
    while (Process32Next(hProcessSnap, &pe32)) {
        pstree.addProc(pe32.szExeFile, pe32.th32ProcessID, pe32.th32ParentProcessID);
        numOfProc++;
    }

    // 출력 예제에 맞춰 출력
    puts(" ########### process tree ############");
    printf("Number of Running Processes = %d\n", numOfProc);
    pstree.printPSTree();

}
```

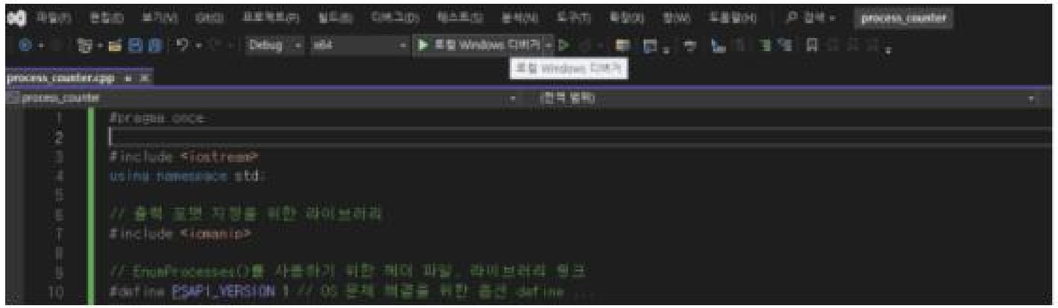
Test description



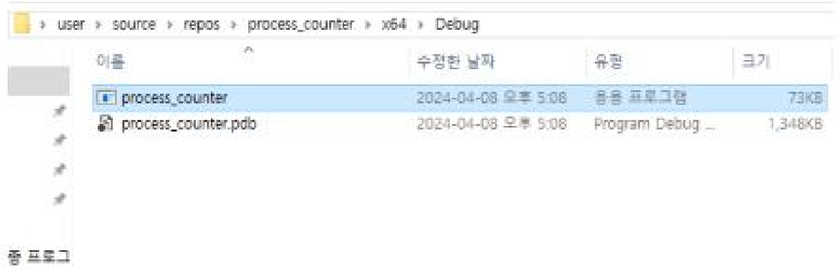
User documentation

1. .exe 파일 실행 (빌드된 상태)

2. 로컬 디버거로 실행 : 프로젝트 생성 후 소스 파일에 추가 후 로컬 디버거로 실행



3. 빌드 후 .exe 파일 실행 : 마찬가지로 프로젝트 생성 후 빌드 - 솔루션 빌드 후 .exe 파일을 찾아 실행



Self-evaluation

85/100

- 헤더 파일을 나눠서 더 모듈화 할 수 있었으나 편의성의 문제로 분리하지 않음. 실습해보면 좋았을 것 같음.
- 클래스나 구조체 등은 적절히 잘 활용하였음