하위 설계

2316 황윤서

본 문서에서는 2020년 리눅스 시스템 프로그래밍 프로젝트를 위한 하위 설계를 기술한다. 하위 설계에서는 S/W 블록 구현을 위한 S/W 모듈(함수)을 설계하고, 각 S/W 모듈(함수)의 슈도코드를 기술하고, S/W 블록 기능 확인을 위한 Test Case를 기술한다.

1. S/W 블록 정보 획득

1.1. S/W 블록 정보 저장

- S/W 블록 정보를 저장하고 있는 파일로부터 S/W 블록 정보 저장용 구조체 배열에 저장한다.
- S/W 블록 정보 저장을 위한 구조체는 swName, swArgv를 포함하여 저장한다.
- struct SwInfo

```
{
    char swName[256];
    char swArgv[4][256];
};
```

1.2. S/W 블록 정보 획득 함수

- 파일로부터 S/W 블록 정보를 획득한다.
  파일에 기록된 한 줄 마다 S/W 블록 정보가 기술되어 있고
  하나의 S/W 블록 정보에 대한 S/W Name, Argument는 “;“로 구분되어 있으므로, ”;“로 토큰화한 후 공백을 삭제하여 저장한다.
- 슈도코드vim

```
void readFileList(int* swBlockCount, struct SwInfo* pSwInfos)
{
    while( !EOF )
    {
        // FileList File로부터 S/W 정보를 한 줄 씩 획득한다.
        // 획득한 S/W 정보를 ”;“을 token으로 하여 분리한다.
        // 공백을 삭제한 후 swInfo structure array에 저장한다.
        // swBlockCount를 1 증가시킨다.
    }
}
```

2. S/W 블록 초기화

2.1. S/W 블록 초기화 정보 저장

- S/W 블록을 초기화 할 때마다 초기화된 S/W 블록 정보, 초기화 시간, 초기화 사유, 초기화 회수 등을 S/W 블록 초기화 정보 저장용 구조체 배열에 저장한다.
- S/W 블록 초기화 정보 저장용 구조체에는 swInfo, startTime, status, reason, restartCount를 포함하여 저장한다.
- struct SwStartInfo

```
{
    struct SwInfo swInfo;
    pid_t swPID;
    time_t startTime;
    int status;
    int restartCount;
    int needStart;
}
```

2.2. S/W 블록 Status 문자열 변환 함수

- status 값을 매개변수로 받으면 문자열로 바꿔 반환한다.
- 슈도코드

```
char* convertStatusToString(int status)
{
    if (status == -9999) {
        // status값이 초기값인 -9999이면
        // "Init"을 반환
    }
    if (WIFEXITED(status))
    {
        // status 값이 정상종료된 프로세스의 것이면
        // "Exit(" + WEXITSTATUS(status) + ")"를 반환
    } else if (WIFSIGNALED(status))
    {
        // status 값이 signal로 종료된 프로세스의 것이면
        // "Signal(" + WTERMSIG(status) + "," +
        // strsignal(WTERMSIG(status)) + ")"를 반환
    }
}
```

2.3. S/W 블록 초기화 시간 문자열 변환 함수

- S/W 블록 초기화 시간을 매개변수로 받아 "2020.06.30. 15:20:33"와 같은

형식으로 문자열 변환하여 반환한다.

- 슈도코드

```
char* convertTimeToString(time_t startTime)
{
    // “2020.06.30. 15:20:33“ 형식으로 문자열 변환하여 반환한다.
}
```

2.4. S/W 블록 초기화 로그 기록 함수

- S/W 블록 초기화 정보를 저장하는 SwStartInfo structure를 매개변수로 받아 ./log/restart.txt에 출력한다.
- 슈도코드

```
int writeSwStartLog(struct SwStartInfo swStartInfo)
{
    // swStartInfo안에 있는 내용 중 swInfo 구조체 멤버변수의
    // swName멤버변수,
    // swStartInfo의 startTime 멤버변수 값을 2.2에서 정의한
    // convertTimeToString함수로 문자열로 바꾼 값,
    // status 멤버변수값을 2.3에서 정의한 convertStatusToString함수로
    // 문자열로 바꾼 값, restartCount 값을 ./log/restart.txt에 출력한다.
}
```

2.5. S/W 블록 초기화 함수

- 인자로 S/W 블록의 이름과 argument들을 받아 S/W 블록을 초기화하고, S/W 블록의 pid 값을 반환한다.
- 슈도코드

```
int swForkAndExec(const char* swName, char argv[4][256])
{
    // 현재 경로 문자열을 temp 변수에 저장한다.
    // 현재 프로세스를 포크하여 자식 프로세스를 생성한다.
    if (pid == 0) {
        // temp 문자열 뒤에 실행파일들이 있는 ”/bin/“과
        // swName을 덧붙인다.
        // execl 함수를 통해 temp 경로에 있는
        // swBlock을 argv와 함께 초기화시킨다.
    } else if (pid > 0) {
        return pid;
    }
    return -1;
}
```

2.6. 모든 S/W 블록 초기화 함수

- 인자로 받은 SwInfo structure array에 포함된 모든 S/W 블록을 초기화하고, 인자로 받은 SwStartInfo structure array를 초기화한다.
- 슈도코드

```
int swInitStart(int swBlockCount, struct SwInfo* swInfos, struct SwStartInfo* swStartInfos)
{
    for(i = 0; i < swBlockCount; i++)
    {
        // 2.5에서 선언한 swForkAndExec 함수로
        // SwInfo structure의 I번째 인덱스에 해당하는
        // S/W 블록을 Argument와 함께 초기화시킨다.
        pid = swForkAndExec(...생략);
        if (pid < 0) return -1;
        // SwStartInfo structure array의 I번째 인덱스에 해당하는
        // 요소의 swInfo 멤버변수에 SwInfo structure array의
        // I번째 인덱스에 해당하는 S/W 블록 정보를,
        // pid 값을 swPID 멤버변수에,
        // 현재 시간을 startTime 멤버변수에,
        // 초기값으로 -9999을 status 멤버변수에,
        // 1을 restartCount 멤버 변수에 대입한다.
        // 2.2에서 정의한 writeSwStartLog 함수를 이용하여
        // SwStartInfo structure array의 I번째 인덱스에 해당하는
        // 요소를 ./log/restart.txt에 기록한다
    }
    return 0;
}
```

3. S/W 블록 재초기화

3.1. 시그널 핸들러 등록 함수

- 자식 프로세스인 S/W 블록 프로세스가 죽었을 때 전달받는 SIGCHLD 시그널을 전달 받았을 때 S/W 블록 재초기화 신호 함수가 실행되도록 SIGCHLD 시그널의 핸들러로 S/W 블록 재초기화 신호 함수를 등록한다.
- 슈도코드

```
int setSignalHandler(void)
{
    // sigaction structure의 sa_mask 값을 빈 시그널 집합으로
    // 만든다.
```

```
        // SIGCHLD 시그널을 전달받았을 때,
        // S/W 블록 재초기화 신호 함수가 실행되도록
        // 시그널 핸들러 함수를 등록한다.
    }
```

3.2. S/W 블록 재초기화 신호 함수(SIGCHLD 시그널 핸들러 함수)

- SIGCHLD 시그널이 전달되었을 때의 시그널 함수로써 SIGCHLD 시그널이 전달되었을 때 실행되며, 시그널을 보낸 자식 프로세스의 pid를 가져와 SwStartInfo structure array 각 요소의 swPID와 비교하여 pid 값이 같은 SwStartInfo structure array 요소의 status 값을 설정하고 needStart 멤버변수로 1로 설정하여 S/W 블록 재초기화 동작이 일어나도록 한다. 또한 전역으로 선언된 gNeedDisplayStartInfo 값을 1로 만들어 S/W 블록 기동 정보 출력 동작이 일어나도록 한다.
- 슈도코드

```
void handler(int signum)
{
    // 죽은 자식 프로세스의 pid와 status 값을 waitpid 함수의 첫 번째
    // 인자로 -1, 세 번째 인자(옵션)으로 WNOHANG을 주어 받아
    // 온다.

    if (pid > 0)
    {
        for (i = 0; I < S/W 블록의 개수; I++)
        {
            if (
                pid ==
                SwStartInfo structure array의 I번째 요소의 swPID
            )
            {
                // SwStartInfo structure array의 I번째 요소의
                // status 멤버 변수에 위에서 받아온 status 값을,
                // needStart 멤버 변수에 1을 대입하여
                // 해당 S/W 블록의 재초기화 동작이 일어나도록 하고
                // break하여 반복문에서 벗어난다.
            }
        }
    }
```

```
    }
```

3.3. S/W 블록 재초기화 함수

- SwStartInfo structure array 각 요소에서 needStart 멤버변수가 1인 S/W 블록을 포인터로 받아 해당 S/W 블록을 재초기화 한다.
- 슈도코드

```
int swRestart(struct SwStartInfo* swStartInfo)
{
    // 2.5에서 선언한 swForkAndExec 함수로
    // 인자로 받은 S/W 블록을 초기화한다.
    pid = swForkAndExec(...생략);
    if (pid < 0) return -1;
    swStartInfo에 pid 및 현재시간 대입 후,
    swStartInfo의 현재 restartCount 멤버변수를 1증가시킨다.
    // 2.2에서 정의한 writeSwStartLog 함수를 이용하여
    // 재초기화 정보를 ./log/restart.txt에 기록한다
    return 0;
}
```

3.4. S/W 블록 체크 및 초기화 함수

- Sw블록 넘버를 인자로 받아 해당 S/W 블록이 초기화가 필요한지 확인하고, 3.3.에서 정의한 swRestart함수로 해당 S/W블록을 초기화한다.
- 슈도코드

```
int swBlockNumCheckAndRestart(int swNum)
{
    if (swStartInfos[swNum].needStart) {
        if (swRestart(...생략) < 0) {
            return -1;
        }
        swStartInfos[swNum].needStart = 0;
        // 초기화 후에는 1을 반환하여 swStartInfos 출력 동작이
        // 일어나도록 한다.
        return 1;
    }
    return 0;
}
```

4. S/W 블록 기동 정보 조회

4.1. S/W 블록 기동 정보 출력 함수

- 인자로 받은 SwStartInfo structure array의 모든 요소를 출력한다. S/W 블

록이 초기화될 때마다 모든 S/W 블록에 대한 기동 정보를 출력한다. S/W 블록 초기화 여부는 전역으로 선언된 gNeedDisplayStartInfo 변수로 결정된다. gNeedDisplayStartInfo변수가 1이면 임의의 S/W블록이 초기화 되어 기동 정보 출력이 필요한 상태를 의미하고 0이면 기동 정보를 이미 출력하여 출력할 필요가 없는 상태를 의미한다.

- 슈도코드

```
int printSwStartInfos(int swBlockCount, struct SwStartInfo* swStartInfo)
{
    for (i = 0; I < swBlockCount; I++)
    {
        // SwStartInfo structure array의 I번째 요소 중
        // swInfo 구조체 멤버변수의
        // swName멤버변수,
        // swStartInfo의 startTime 멤버변수 값을 2.2에서 정의한
        // convertTimeToString함수로 문자열로 바꾼 값,
        // status 멤버변수값을 2.3에서 정의한
        // convertStatusToString함수로 문자열로 바꾼 값,
        // restartCount 값을 한 줄에 출력하고 줄바꿈한다.
    }
    return 0;
}
```

5. Test Case

5.1. S/W 블록 초기화 Test

5.1.1. Test Case 1. Text File Parsing 확인

5.1.1.1. 사전준비

- S/W block 및 S/W 블록 정보 파일을 준비한다.

5.1.1.2. 시험방법

- Application을 실행한다.

5.1.1.3. 결과확인 방법

- Application이 획득한 S/W block 정보를 확인한다.
  (S/W 블록 기동 정보 조회 기능 활용)

5.1.2. Test Case 2. Log File Write 확인

5.1.2.1. 사전준비

- S/W block 및 S/W 블록 정보 파일을 준비한다.

5.1.2.2. 시험방법

- Application을 실행한다.

5.1.2.3. 결과확인 방법

- ./log/restart.txt 파일의 내용을 확인한다.(cat 명령 활용)

5.2. S/W 블록 재초기화 Test

5.2.1. Test Case 1. 재초기화 기능 확인

5.2.1.1. 사전준비

- S/W block 및 S/W 블록 정보 파일을 준비한다.

5.2.1.2. 시험방법

- Application을 실행하고 초기화된 S/W 블록들을 강제로 종료시킨다.

5.2.1.3. 결과확인 방법

- S/W 블록들의 프로세스가 다시 복구되었는 지 확인한다.
  (“ps -al | grep SwBlock“명령 및 ./log/restart.txt에서 restart count 값 확인)