리눅스 환경에서 결함 허용 프로세스를 위한 검사점 및 복구 도구

임 성 락[†] · 김 신 호[††]

요 약

본 논문에서는 장시간 실행이 예상되는 결함 허용 프로세스를 위한 검사점 및 복구 도구를 제시한다. 제시한 도구의 기본 개념은 프로세스의 실행 상태를 주기적으로 저장함으로써 시스템 결함으로 인해 실행이 정지되었을 경우, 결함이 발생하기 전의 실행 상태를 복구하여 계속 실행시키는 것이다. 제시한 도구에서는 검사점 및 복구를 위하여 결함 허용 프로세스의 소스 코드를 수정할 필요가 없다. 이를 위하여 결함 허용 프로세스를 위한 파일명과 검사점 주기를 사용자가 직접 지정하도록 설계하고, 두 개의 시스템 호출(save, recover)을 추가하였다. 마지막으로 제시한 기법의 타당성을 검토하기 위하여 리눅스 환경(커널 2.4.18)에서 구현하였다.

A Checkpoint and Recovery Facility for the Fault-Tolerant Process on Linux Environment

Seong-Rak Rim[†] · Sin-Ho Kim[††]

ABSTRACT

In this paper, we suggest a checkpoint and recovery facility for the fault-tolerable process which is expected to be executed for a long time. The basic concept of the suggested facility is to allow the process to be executed continuously, when the process was stopped due to a system fault, by storing the execution status of the process periodically and recovering the execution status prior to the fault was occurred. In the suggested facility, it does not need to modify the source code for the fault-tolerable process. It was designed for the user to specify directly the file name and the checkpoint frequency, and two system calls(save, recover) were added. Finally, it was implemented on the Linux environment(kernel 2.4.18) for checking the feasibility.

키워드 : 검사점 도구(Checkpoint Facility), 복구 도구(Recovery Facility)

1. 서 론

결함 허용 프로세스를 위한 검사점(checkpoint) 및 복구(recovery) 기법은 장시간 실행을 요하는 프로세스에서 예상치 못한 시스템 결함으로 인한 피해를 최소화하는데 매우 중요한 도구이다[1, 2]. 이 기법은 후방 오류 복구(backward error recovery) 기법 중의 하나로 프로세스의 실행 상태를 저장해 둠으로써 시스템의 결함이 발생하였을 때, 결함이 발생하기 전까지의 프로세스 실행 상태를 복구하여 프로세스가 계속적으로 실행될 수 있도록 한다. 이때 프로세스의 실행 상태 저장을 검사점이라 하고, 검사점 상태에서부터 계속 실행시키는 것을 복구라고 한다.

일반적으로 검사점 및 복구 도구는 사용자 라이브러리 혹은 커널 수준에서 구현할 수 있다[1, 2]. 전자의 방법은 커널 수정 없이 사용자 수준에서 검사점 및 복구 기능을 지원함으로 사용자의 편의성을 제공할 수 있지만 라이브러리 특성으로 인해 비효율적이다. 반면 후자의 방법은 보다 효율적이지만 커널 수정의 어려움이 있다. [1]에서는 커널 수준의 검사점 및 복구를 위한 시스템 호출을 추가하여 사용자 라이브러리 방법에 대한 커널 수준의 효율성을 제시하였다. 또한 사용자의 투명성을 제공하기 위하여 응용 프로그램 작성 단계에서 사용자가 직접 검사점을 지정하도록 되어 있다. 따라서 임의의 프로세스의 결함 허용을 지원하기 위해선 반드시 그 프로그램의 소스 코드 내부에 검사점 설정이 추가되어야 한다. 뿐만 아니라 응용 프로그램의 소스 코드를 구하는 일은 어려울 뿐만 아니라, 재 컴파일 과정은 상당한 시간을 요하며, 어떤 경우에는 특정한 컴파일 환경을 필요로 한다.

본 논문에서는 이러한 문제점을 해결하기 위한 결함 허

※ 본 연구는 2004년도 호서대학교 벤처산학컨소시엄연구 지원과제로 수행된 연구결과입니다.
† 종신회원 : 호서대학교 컴퓨터학부 교수
†† 준 회 원 : 호서대학교 대학원 컴퓨터응용기술
논문접수 : 2003년 12월 19일, 심사완료 : 2004년 4월 30일

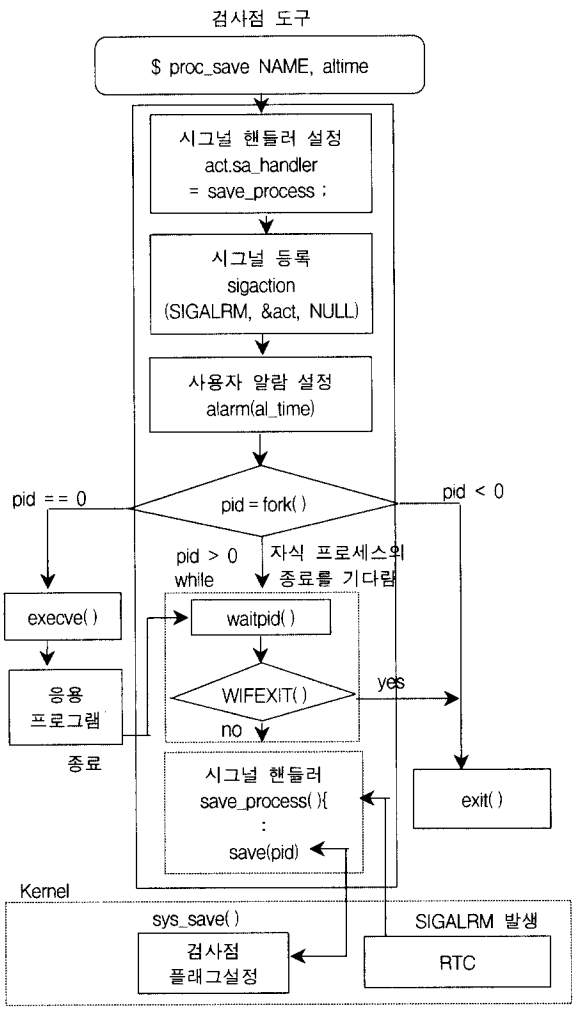
용 프로세스 복구 기법을 제시한다. 제시한 기법의 기본 개념은 임의의 응용 프로그램의 소스 코드를 전혀 수정하지 않고, 오직 실행 파일만을 가진 사용자에게 그 응용 프로그램에 대한 검사점 및 복구를 지원하는 것이다. 제시한 기법의 타당성을 검증하기 위하여 리눅스 커널(2.4.18) 내부에 두 개의 시스템 호출을 추가하고, 사용자 수준에서의 결함 허용 프로세스 검사점 및 복구 도구를 실험한다.

2. 설 계

본 논문에서는 결함 허용 프로세스의 검사점 및 복구를 위하여 사용자 수준의 도구(proc_save, proc_rec)와 시스템 호출(save(), rec())에 대한 커널 함수(sys_save, sys_rec)를 설계한다.

2.1 검사점 도구 : proc_save

결함 허용 프로세스의 실행 상태를 저장하기 위한 검사점 도구는 (그림 1)과 같이 설계한다.



(그림 1) 검사점 도구(proc_save)

검사점 도구 프로세스는 사용자로부터 결함 허용 프로세스를 위한 응용 프로그램 명(NAME)을 입력받아 들인다. 시그널을 이용하여 주기적으로 프로세스의 검사점 플래그를 설정하기 위해서 시그널 핸들러를 설정하고 SIGALRM을 등록 시킨후 사용자로부터 입력받은 검사점 주기로 alarm 시간을 설정한다. 부모 프로세스(proc_save)는 새로운 프로세스를 생성하고, 입력받은 파일명에 해당되는 응용 프로그램을 실행(execve())시킨다.

자식 프로세스(응용 프로그램)가 실행되는 동안 커널에서 주기적으로 시그널(SIGALRM)이 발생하면 부모 프로세스에 존재하는 시그널 핸들러(save_process())가 수행된다. 시그널 핸들러는 추가된 시스템 호출인 save()를 호출하여 커널 내의 sys_save()를 수행한다. sys_save()에서는 인자로 넘어온 pid를 이용하여 자식 프로세스의 검사점 플래그를 설정한다. 자식 프로세스의 실행 상태 저장은 사용자 모드로 복귀할 때 이루어지도록 한다.

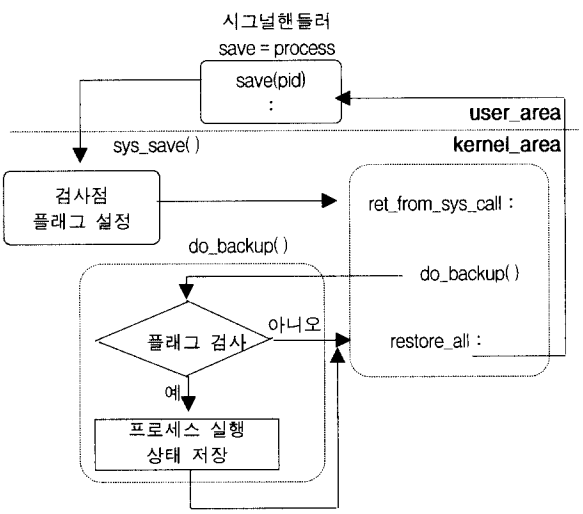
한편 모든 프로세스들이 커널 모드에서 사용자 모드로 복귀할 때, ret_from_sys_call()에서 호출되는 do_backup()에서 플래그가 설정이 되어있는지를 확인하는 과정을 추가함으로써 검사점 플래그가 설정되지 않은 프로세스들에 대한 실행 상태 저장은 생략하도록 한다.

시그널 핸들러는 save()를 수행 후 사용자가 부여한 주기(TIME) 값으로 alarm()을 설정한다.

이와 같은 방법으로 결함 허용 프로세스의 검사점 설정 및 실행 상태의 저장이 주기적으로 이루어지도록 한다.

2.2 시스템 호출 : save( ) 함수

시스템 호출 save()는 프로세스의 상태를 저장하는 기능을 수행한다. 프로세스의 실행 상태를 저장하기 위한 시스템 호출의 커널 내부 함수(sys_save())는 (그림 2)와 같이 설계한다[1].



(그림 2) sys_save( ) 함수 구조

(그림 2)에서 sys_save()은 단순히 검사점 플래그를 설정한다. 프로세스의 실행 상태 저장은 나중에 커널 모드에서 사용자 모드로 전환하기 위한 루틴(ret_from_sys_call)에 do_backup()을 추가함으로써 가능하도록 한다.

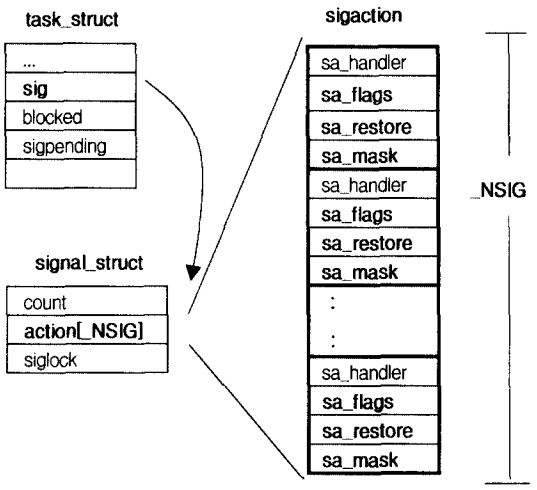
do_backup()에서는 현재 실행 중인 프로세스의 검사점 설정 플래그를 검사하여 플래그 값이 설정되어 있을 경우에만, 프로세스의 실행 상태를 저장하기 위한 "[실행파일명].bak"이라는 복구 파일을 생성하고 복구를 위한 정보를 저장한다. 프로세스의 실행 상태를 복구하기 위하여 다음과 같은 정보들이 요구된다.

2.2.1 프로세스의 구조체 : task_struct

일반적으로 커널 내부에서는 프로세스의 모든 정보를 관리하기 위하여 프로세스 구조체(task_struct)를 유지한다[3, 4]. do_backup()에서 결함 허용 프로세스의 검사점 플래그 설정을 위한 필드를 프로세스 구조체에 추가한다.

2.2.2 프로세스 신호 정보 : signal_struct

결함 허용 프로세스가 시스템 오류에 의해 종료될 경우 이에 대한 신호 정보를 저장한다. 프로세스의 신호 정보(signal_struct)는 (그림 3)과 같은 구조로 프로세스 구조체에 연결되어 있다[1, 5].

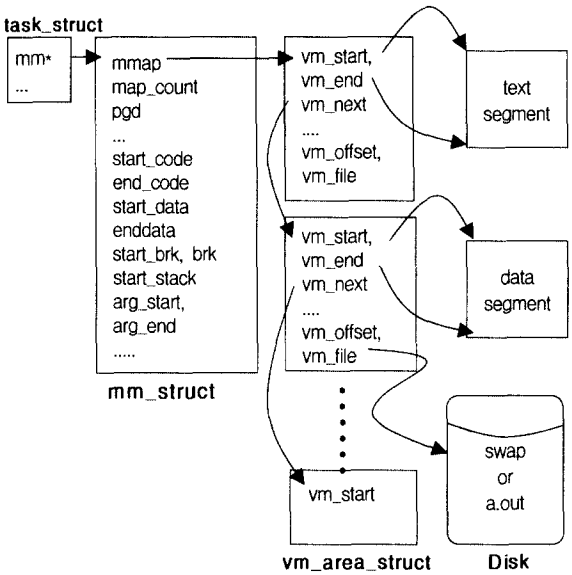


(그림 3) signal_struct 연결 구조

2.2.3 프로세스의 주소 공간 : mm_struct

결함 허용 프로세스 복구를 위해 필요한 프로세스의 주소 공간(mm_struct)은 (그림 4)와 같은 구조로 프로세스 구조체에 연결되어 있다.

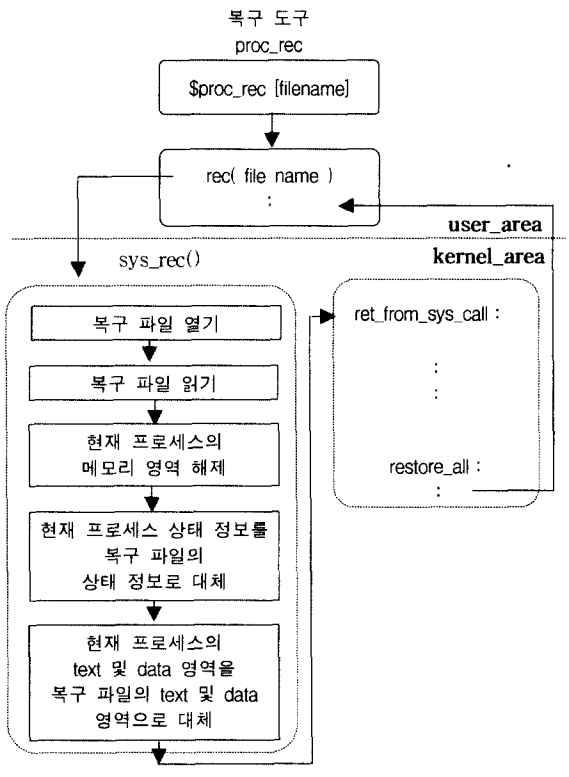
결함 허용 프로세스의 복구를 위하여 프로세스의 text 및 data segment의 내용을 저장할 필요가 있다. (그림 4)에서 프로세스의 주소 공간과 관련된 정보는 프로세스 구조체 내부의 mm_struct 자료구조와 mm_struct 구조체의 mmap 포인터가 가리키는 vm_area_struct 리스트에 있다[6]. vm_area_struct는 프로세스의 가상 메모리 영역으로써 여러 개의 segment로 구성되며, 결함 허용 프로세스의 복구를 위한 text 및 data segment가 저장된 메모리 주소는 vm_area_struct 구조체의 vm_start와 vm_end가 각각 가리킨다[1, 7].



(그림 4) mm_struct 연결구조

2.3 복구 도구 : proc_rec

결함 허용 프로세스의 실행 상태를 복구하기 위한 복구 도구는 (그림 5)와 같이 설계한다.



(그림 5) 복구 도구(proc_rec)

복구 도구 프로세스는 검사점 설정으로 저장된 복구 파일명(NAME.bak)을 입력으로 받아들여 단순히 결함 허용 프로세스의 복구를 위한 시스템 호출함수(rec())를 호출한다.

2.4 시스템 호출 : rec( ) 함수

시스템 호출 rec()는 프로세스의 상태를 복구하는 기능을 수행한다. 프로세스의 실행 상태를 복구하기 위한 시스템 호출의 커널 내부 함수(sys_rec())는 (그림 5)와 같이 설계한다.

sys_rec()는 복구 파일로부터 읽어 들인 결함 허용 프로세스의 상태 정보와 text 및 data 영역의 내용을 읽어 들인 후, 현재 실행 중인 복구 프로세스의 상태 정보와 text 및 data 영역의 내용을 복구 파일로부터 읽어들인 결함 허용 프로세스의 상태 정보와 text 및 data 영역으로 대체한다.

3. 구현 및 실험

본 논문에서는 두 개의 시스템 호출과 사용자 수준에서의 프로세스 상태 저장 도구 이용한 결함 허용 프로세스의 복구 기법을 제시하였다. 제시한 기법의 타당성을 검토하기 위하여 리눅스 커널 2.4.18 내부에 두 개의 시스템 호출을 구현하고, 이를 이용한 결함 허용 프로세스의 복구 기법을 실험하였다.

3.1 proc_save 구현

```
                    :
/* 입력으로 받아들일 인자를 설정 */
        if(argc != 3)
        {
fprintf(stderr, "command parameta : <proc_save>
<program_name> <time> ₩n") ;
        }
                    :
/* 시그널 핸들러 설정 */
act.sa_handler = save_process ;
/* 마스크에대한 시그널 셋의 모든 시그널들을 제거 */
sigemptyset(&act.sa_mask) ;
/* 시그널 핸들러 실행시 옵션을 설정 */
act.sa_flags = SA_RESTART ;
/* SIGALRM 시그널 핸들러를 등록한다. */
if( sigaction(SIGALRM, &act, NULL) < 0)
                    :
/* 알람 초기화 */
alarm(al_time) ;
/* 새로운 프로세스를 생성 */
pid = fork() ;
if(pid < 0) fprintf(stderr, "fork error ₩n") ;
if(pid == 0)            /* child process */
{
        printf("child processing ₩n") ;
        /* 대상 응용프로그램 실행 */
        if(execve(filename, 0, 0) < 0)
                    :
}
else                    /* parent process */
```

```
while(1)
{
        /* 자식프로세스의 종료를 기다림 */
        waitpid(pid, &status, 0) ;
        /* 자식 프로세스가 정상 종료 되었는지 확인 */
        if(WIFEXITED(status))
                :
        else
        {
                printf("pause ₩n") ;
        /* 시그널이 발생할 때까지 프로세스를 블록 시킴 */
                pause() ;
        }
}
/* 시그널 핸들러 */
static void save_process(int i)
{
                :
printf("₩ncurrent save_process processing number : %d₩n", k++) ;
        /* 자식프로세스를 멈춤 */
        kill(pid, SIGSTOP) ;

/* 자식 프로세스의 상태를 확인 */
if(waitpid(pid, &status, WUNTRACED) < 0)
        perror("SIGSTOP error") ;
                :
        /* save시스템 호출 */
        save(pid) ;
        /* 알람 초기화 */
        alarm(0) ;
        /* 알람 재설정 */
        alarm(al_time) ;
                :
        printf("kill SIGCONT ₩n") ;
        /* 자식 프로세스의 작업을 재시작 */
        kill(pid, SIGCONT) ;
                :
```

3.2 sys_save( ) 구현

결함 허용 프로세스의 실행 상태를 저장하기 위한 시스템 호출(save())을 지원하기 위하여 다음과 같이 커널 내부 함수들을 수정 및 추가하였다.

3.2.1 검사점 플래그 추가 및 설정

결함 허용 프로세스를 식별하기 위한 검사점 플래그(int need_bak)를 task_struct에 추가하고, 인자로 받은 대상 프로세스에 대한 검사점 플래그 값을 TRUE로 설정하는 sys_save()를 정의한다.

```
struct task_struct {
        :
    int need_bak ; /* 검사점 플래그 추가*/
} ;
sys_save() {
    /*검사점 플래그 추가*/
      current → need_bak = TRUE ;
}
```

3.2.2 ret_from_sys_call 수정

시스템 호출로부터 사용자 모드로 복귀하는 루틴(ret_

from_sys_call)에 프로세스 실행 상태 정보 저장을 위한 do_backup()을 다음과 같이 추가한다.

```
ENTRY(ret_from_sys_call)
        :
        jne signal_return
        call SYMBOL_NAME(do_backup) /* do_backup() 추가 */
restore_all :
        RESTORE_ALL
```

3.2.3 do_backup() 추가

프로세스 실행 상태 정보 저장을 위하여 다음과 같은 기능을 수행하는 do_backup() 추가한다.

```
do_backup() {
    make_file() ; /*복구 파일명 부여 및 생성*/
    write_file() ; /*프로세스의 실행 상태 저장*/
}

make_file() {
   char *extend = ".bak" ;
        memcpy(filename, current → comm, k) ;
        filename [i++] = extend [i++] ;
        filp_open(filename, O_CREAT, ···) ;
            :
}

write_file() {
        write(file, (char*)current, ···) ;
        write(file, (char*)current → sig, ···) ;
        write(file, (char*)&vm_area_count, ···) ;
        write(file, (char*)current → mm, ···) ;
        write(file, (char*)regs, ···) ;
        write(file, (char*)current → mm → mmap, ···) ;
        write(file, (char*)current → mm → mmap → vm_start,) ;
            :
}
```

3.3 sys_rec( ) 구현

결함 허용 프로세스의 실행 상태를 복구하기 위한 시스템 호출(rec())을 지원하기 위하여 다음과 같이 정의된 커널 함수들을 추가하였다.

```
sys_rec() {
        :
   filename = getname(name) ; /*파일명 가져오기*/
   filp_open(filename, O_RDONLY, 0600) ; /*파일열기*/
/* 복구 상태 정보를 읽기 */
   read(file, (char*)&task, n, &file → f_pos) ;
   read(file, (char*)&sig, n, &file → f_pos) ;
   read(file, (char*)&vm_area_count, n, &file → f_pos) ;
   read(file, (char*)&mm, n, &file → f_pos) ;
   read(file, (char*)&regs, n, &file → f_pos) ;
/* 현재 프로세스의 메모리 영역 해제 */
   flush_old_exec(&bprm) ;
/* 결함 허용 프로세스의 상태 정보 복구 */
   restore_task(&task) ;
   restore_mm(&mm) ;
   restore_sig(&sig) ;
/* 결함 허용 프로세스의 text 및 data 복구 */
   read(file, (char*)&tmp, n, &file → f_pos) ;
   do_mmap(NULL, tmp, vm_start, ···) ;
   find_vma(current → mm, tmp.vm_start) ;
   read(file, (char*)tmp.vm_start, n, &file → pos) ;
   putname(filename) ;
}
```

3.4 응용 프로그램 및 복구 실험

본 논문에서 제시한 결함 허용 프로세스의 타당성을 검토하기 위하여 save() 및 rec()를 리눅스 커널 2.4.18 내부에 추가하고 이를 이용한 다음과 같은 응용 및 복구 도구를 작성하여 실험하였다.

• 응용 프로그램 : test.c

```
        /* 계산 수행 */
        for (i = 0 ; i < 100 ; c++) {
                for (j = 0 ; j < 100 ; j++) {
                array[ i ][ j ] = i + j ;
                printf("%d\n", array[ i ][ j ]) ;
                }
        }
```

• 복구 도구 : pro_rec.c

```
        /* rec 시스템호출의 정의 */
        _syscall1(int, rec, char*, name) ;
        int main(int argc, char** argv) {
                    :
        /* rec() 시스템 콜 함수 호출 */
                rec(argv[1]) ;
                return -1 ;
        }
```

3.4.1 결함 허용 프로세스 저장 도구 실행

```
[root@localhost save]# ./proc_save test 1
test    test.c    proc_rec.c  proc_rec    proc_save
```

3.4.2 검사점 파일 생성 및 확인

응용 프로그램(test)이 실행되는 중간에 "ctrl+C"로 비정상 종료시킨 후, 검사점 파일(test.bak)이 생성됨을 확인한다.

```
[root@localhost save]# ls
test    test.c    test.bak    proc_rec.c    proc_rec    proc_save
```

3.4.3 결함허용 프로세스 복구

복구 프로그램인 "proc_rec"로 비정상적으로 종료되었던 결함 허용 프로세스를 복구시킨다.

```
[root@localhost save]# ./proc_rec test.bak
```

4. 결 론

프로세스의 복구 기법은 장시간 실행을 요하는 프로세스

에서 하드웨어, 소프트웨어적 결함으로 발생하는 심각한 피해를 최소화하기 위하여 절대적으로 요구된다. 프로세스 복구를 위한 대부분의 기존기법들은 임의의 프로세스의 결함 허용을 지원하기 위해서 반드시 그 프로그램의 소스 코드 내부에 검사점 설정이 추가되어야만 했다.

본 논문에서는 소스코드 내부에 검사점을 설정하거나 재컴파일을 해야만하는 기존 기법들의 단점을 해결하기 위해 오직 실행파일 만으로도 검사점 및 복구 작업을 수행할 수 있는 사용자수준의 도구를 설계하였다. 제시한 기법은 사용자의 편의성과 각기 다른 시스템 상 에서의 효율성을 극대화 시키기 위해 검사점을 수행할 프로그램을 사용자가 직접 선택할 수 있도록 개선하였으며, 사용자가 응용프로그램의 검사점 주기를 직접 입력할 수 있도록 구현하였다.

참 고 문 헌

[1] 조유근, 최종무, 홍지만 저, "리눅스 매니아를 위한 커널 프로그래밍", 교학사.

[2] 홍지만, 한상철, 윤진혁, 박태순, 염헌영, 조유근, "UnixWare 커널 수준의 효율적인 검사점 및 복구 도구", 정보과학회 춘계 학술발표 논문.

[3] Daniel P. Bovet and Marco Cesati, "Understanding the LINUX KERNEL," O'Reilly.

[4] Keith Haviland, Dina Gray, Ben Salama 저, "UNIX system programing," 홍릉 과학 출판사.

[5] 권상호, 고성규, 강호성, 민기획 저, "Unix & Linux C Programming," 영진닷컴.

[6] Uresh Vahalia 원저 조유근 역, "UNIX의 내부," 홍릉과학 출판사.

[7] 안성진, 정진욱, 박진호, 안용학, 공저, "LINUX 프로그래밍 기술", 양서각.

임 성 락

e-mail : srrim@office.hoseo.ac.kr

1979년 서강대학교 전자공학(학사)

1983년 서울대학교 컴퓨터공학(석사)

1992년 서울대학교 컴퓨터공학(박사)

1983년~1990년 금성반도체(주) 연구소

1993년~현재 호서대학교 컴퓨터학부 교수

관심분야 : 운영체제, 임베디드 시스템

김 신 호

e-mail : jerry_builder@hotmail.com

2003년 호서대학교 뉴미디어(학사)

2003~현재 호서대학교 벤처전문대학원 컴퓨터응용기술(석사)

관심분야 : 운영체제, 임베디드 시스템