□특 집□

멀티프로세서 시스템에서 실행시간 오류복구에 관한 연구

한국전자통신연구소 임 기 욱*

목 차

Ⅰ. 서 론

멀티프로세서 시스템에서의 고장은 전체 시스템을 동작 불능의 상태나 오동작의 상태로 만든다. 특히 실시간 응용 프로그램에서의 프로세서의 고장은 귀중한 인명과 막대한 재산에 피해를 준다. 일반적으로 시스템의 결함 허용을 위해서는 여분(redundant)의 하드웨어나 소프트웨어 모듈의 공급이다. 그러나, 멀티프로세서 시스템에서 프로세서 고장시 오류복구를 위한 여분의 프로세서의 공급은 시스템 오버헤드를 가중 시킨다.

따라서 멀티프로세서 시스템 상에서 프로세서의 고장시 시스템 오버헤드를 최소한으로 줄이고 능률적으로 오류 복구를 할 수 있는 능력을 멀티프로세서 시스템에 부여할 수 있는 실행 시간(run-time) 오류 복구 기법의 개발이 필요하다. 이와 같은 오류 복구 기법이 갖추어야 할 바람직한 특성은 다음과 같다.

• 독립성(independency) : 정상적인 시스템 수행에 미치는 영향이 없어야 한다.

• 완전성(completeness) : 발생된 오류는 반드시 감지되어야 한다.

• 유동성(flexibility) : 같은 오류라도 경우에 따라 처리가 달라질 수 있어야 한다.

• 확장성(extendability) : 새로운 오류에 대한 처리가 추가될 수 있어야 한다.

• 강건성(robustness) : 예상치 못한 오류가 발생해도 시스템이 오동작하거나 전체 시스템의 failure가 일어나서는 안된다.

이와 같은 특성을 완전하지는 않지만 충족히 갖추기 위해 다음과 같은 기능이 실행시간 오류 복구 기법에서 고려되어야 한다.

(1) 결함감지

항시 또는 주기적으로 시스템 상태를 분석하여 결함이 존재하는지의 여부를 감지한다.

(2) 결함 장소 검출

결함이 발생 하였을시 이를 감지하고 결함의 원인이 무엇이며 어느 부분에서 발생하였는지를 알아낸다.

(3) 사용자 정의 복구 행위(recovery action)

* 종신회원

지원

발생된 결함의 종류에 따라 적절한 복구행위를 취할 수 있도록 다음과 같은 기능을 지원한다.

• 상태보고 : 현재 멀티프로세서 시스템의 상태를 주기적으로 또는 필요시 시스템의 상태를 운용자에게 보고한다.

• 오류통지 : 결함 감지시 결함의 발생을 운용자에게 통지한다.

• 행위수행 : 발생시 결함을 처리하기 위하여 시스템에 미리 정의된 행위나 운용자가 명령한 행위를 수행한다.

이와 같은 기능을 갖춘 기법의 효용성을 검증하기 위해 사건처리기라 명명된 도구가 설계되었다. 이 사건 처리기는 4계층(멀티 프로세서 시스템 응용계층, 오류관리 계층, 시스템 상태 정보 관리 계층과 멀티 프로세서 시스템 관리 계층)으로 설계되었으며 이중 가장 핵심은 오류 관리 계층이다. 현재 오류 관리 계층을 중심으로 한 사건 처리기의 prototype이 거의 구현 단계에 있다.

본 고의 내용을 살펴보면 II장에서는 기존의 결함 허용 기법 및 실행시간 결함 진단 기법을 간단히 살펴보고 현재 개발된 오류 복구 기법과 설계된 사건 처리기를 기술하였다. III장에서는 현재 거의 구현 완료된 사건 처리기 prototype의 구조 및 기능을 간단히 기술하였고, 마지막으로 IV장에서는 현재까지의 연구진행과 앞으로의 남은 과제에 대해 요약하였다.

II. 실행시간(Run-time) 오류복구 기법

2.1 실행 시간 결함 진단 기법

멀티프로세서 시스템은 시스템 자체 내에 여분의 부품이 내재되어 있어 이를 활용하여 결함이 존재할 경우에도 정상적인 수행을 할 수 있도록 하는 것이 가능하다. 이와 같은 기능을 실행시간 중에 부여하기 위해서는 결함 모듈을 시스템 스스로 찾을 수 있도록 하는 것이 바람직하다. 이러한 결함 모듈을 스스로 찾는 과정을 결함 진단(fault diagnosis)이라 한다.

지금까지 연구되어 왔던 결함 진단 분야에서는 결함 진단을 위한 모델을 설정하여 이 모델을 기반으로 결함 진단 알고리즘이 개발되었다. 지금까지 개발되어온 결함 진단 모델은 deterministic모델, graph 모델, 확률 모델 등이 있는데 이들을 간략히 살펴보면 다음과 같다.

(가) Deterministic 모델

결함 유형을 미리 결정하고 결정된 결함 유형 각각을 감지할 수 있는 test 입력을 미리 설정한 후 이 test 입력을 이용하여 실행 중에 결함을 감지하도록 하는 모델이다. 이 모델 부류에는 결함 유형과 그에 대한 test 입력을 테이블을 이용하여 표현하는 모델과 Boolean expression을 이용하여 표현하는 모델이 있다.

(나) Graph 모델

시스템은 unit 단위로 구분되며 각 unit가 다른 unit들의 결함 진단 및 진단결과 분석을 행할 수 있다는 가정하에 결함 진단 시스템을 방향서 그래프로 표현하도록 한 모델이다. 이 방향성 그래프는 결함-test 상관 관계와 시스템에서 결함이 unit간에 전달되는 과정을 표현한다.

(다) 확률 모델

Deterministic 모델에 확률을 가미하여 설정한 모델로서 deterministic 모델보다 효율적인 결과를 얻을 수 있으나 각각 결함 유형에 대한 발생 확률을 정확히 산출하는 것이 중요하다.

위의 모델을 토대로 여러가지 알고리즘이 중앙집중식이나 분산처리식으로 개발되었다.

이 점에서 언급한 결함 진단 모델과 알고리즘을 실행시간 중에 적용할 경우 다음과 같은 문제점이 발생할 수 있다.

• 결함 진단을 하기 위해 test입력을 주기적으로 입력시켜 그 출력을 분석하거나 진단시스템을 정상적인 시스템 수행과 interleaving 방식으로 동작시켜야 한다. 이때 이에 따르는 overhead가 정상적인 시스템 수행에 영향을 미칠 수 있다.

• 결함 진단 시스템의 수행시간을 단축하도록 하드웨어로 구현하는 것도 고려해 볼 수 있으나 이 경우 진단 시스템 하드웨어가 기존의 구조에 부가되도록 기존의 구조를 변경하여야 하며 여전히 overhead가 존재한다.

• 위의 모델들은 결함 진단을 위한 것으로 결함으로부터의 복구는 고려되지 않았다. 실제로는 진단 후 결함의 존재가 발생되었을 시, 이 결함의 파급효과를 최소화하고 결함으로부터의 신속한 복구가 필요하나, 이 모델들에서는 이러한 면이 고려되지 않았다.

2.2 제안된 오류 복구 기법

시스템 실행 시간 중에 사용된 오류 복구 기법의 바람직한 특성은 다음과 같다.

• 오류 감지나 복구시 정상적인 시스템 수행에 미치는 영향이 없어야 한다.
• 오류가 영구적인 것이 아닌 일시적인 것일지라도 발생된 오류는 반드시 감지되어야 한다.
• 같은 오류에 대한 처리 방식이 경우에 따라 달라질 수 있으므로 오류 기법 자체에 유동성(flexibility)이 있어야 한다.
• 설계시 미처 고려되지 못한 오류가 발견될 수 있으므로 확장성(extendability)이 있어야 한다.
• 고려되지 못한 오류의 발생시 시스템이 오동작을 하거나 전체 시스템의 failure가 일어나지 않도록 하는 기능이 오류 복구 기법에 고려되어야 한다.(robustness)

위의 특성을 완전히 충족시키는 오류 복구 기법이 이상적이나 현실적으로는 실현 가능성이 없으므로 위의 특성을 될 수 있는대로 충분히 충족시킬 수 있는 오류 복구 기법을 개발하는 것이 필요하다. 이러한 오류 복구 기법은 하드웨어에서의 긴밀한 협조가 요구되나 현실적으로 적용가능하도록 기존의 하드웨어 변경없이 효율적인 실행시간 오류 복구를 할 수 있도록 운영체제 상에 적용할 수 있는 기법을 다음과 같은 특성을 갖도록 개발하였다.

• 오류 감지나 복구시 정상적인 시스템 수행에 미치는 영향을 될 수 있는 대로 최소화한다.
• 일시적 오류도 될 수 있는 대로 감지할 수 있도록 한다.
• 유동성(flexibility)이 있다.

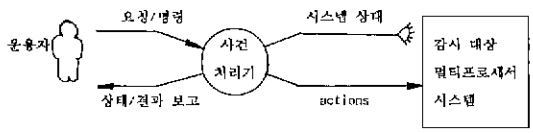

그림 1. 실행 시간 오류 복구 기법의 기본 개념

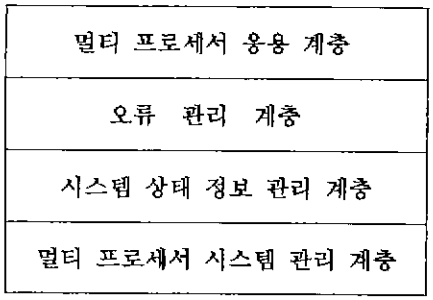

그림 2. 사건 처리기의 구조

• 확장성(extendability)이 있다.
• 최소한의 견고성(robustness)이 주어진다.

이 기법의 기본 개념은 그림 1에서 보듯이 멀티프로세서 시스템에서 오류가 발생되는지를 항상 감시하고 있다가 오류의 발생이 감지되면 그 오류에 대해 미리 정의된 시스템 처리 행위(action)를 취하거나 운용자에 의해 주어진 행위(action)를 취함으로서 발생된 오류의 파급효과를 최소화하거나 그 오류로부터 복구를 하는 것이다.

그림 1에서 사건 처리기의 역할은 기본적으로 다음의 3가지 기능을 수행하고 추가적인 다른 기능을 수행할 수도 있다.

• 상태보고 : 주기적으로 또는 필요시 시스템의 상태를 운용자에게 보고한다.
• 오류통지 : 오류의 발생을 운용자에게 통보한다.
• 행위수행 : 오류의 발생시 시스템에 미리 정의된 행위나 운용자에 의해 주어진 행위를 취한다. 감지된 오류에 대한 행위가 정의되어 있지 않을시 운용자에게 통보하고 그에 대한 처리 행위를 요청한다.

이 사건 처리기는 본 연구에서 제안된 실행시간 오류기법의 핵심이며 기본 구조는 그림 2에 있듯이 4개의 계층으로 나누어 볼 수 있다.

이와 같이 계층 구조를 적용한 것은 위 계층이 아래 계층의 기능을 활용함으로써 각 계층의 복잡도를 줄이고 시스템 구현이 용이토록 하기 위함이다.

(가) 멀티 프로세서 응용 계층

멀티 프로세서 운영 체제에서 지원하는 interprocess communication, resource allocation, memory management, 사용자의 login 또는 logout, 각 프로세서의 work load 등 운영체제에서의 시스템 상태 정보 수집 기능 등이 이 계층에 속한다. 기본적으로는 기존의 운영체제가 가지고 있는 기능들이다.

(나) 시스템 상태 정보 관리 계층

수집된 상태 정보를 보관하고 오류 관리 계층에서 이용될 수 있도록 관리하는 계층이다. 또한 운영체제가 수집하지 않은 상태 자료가 필요한 경우 그 자료를 수집하는 역할도 한다. 상태 정보를 위한 상태 정보 저장소(Status Information Base: SIB)의 관리도 이 계층의 역할이다. SIB의 구조나 관리는 매우 중요하나 현재 이에 대한 연구가 미흡한 실정이고 앞으로 행하여야 할 중요한 연구과제 중의 하나이다.

(다) 오류관리 계층

사건 처리기의 핵심 계층으로서 다음과 같은 기능을 수행한다.

- 운용자와의 interaction : 운용자는 멀티프로세서 응용 계층상의 응용 프로그램일 수도 있다.
- 오류의 감지 : 상태 정보로부터 오류의 발생여부를 분석한다.
- 행위수행 : 시스템에 의해 또는 운용자에 의해 주어진 행위를 수행한다.

(라) 멀티 프로세서 시스템 관리 계층

멀티 프로세서 시스템에서 수행되는 정상적인 응용 프로그램에 대한 계층으로써 아래 계층에서 주어지는 기능을 충분히 활용하도록 하는 것이 바람직하다.

본 고에서는 위의 계층 중 멀티 프로세서 시스템 관리 계층과 시스템 상태 정보관리 계층의 기능을 기존의 운영체제 기능을 활용하여, 오류관리 계층의 기능에 관한 설계를 다음 장에서 소개한다.

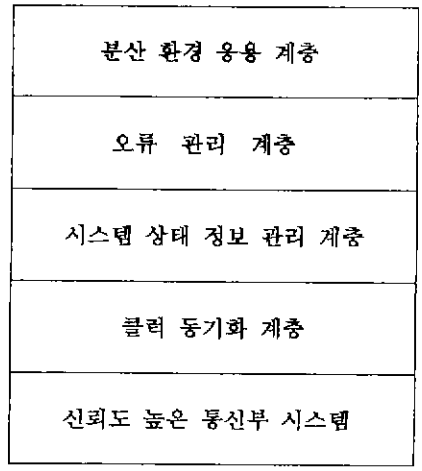

그림 3. 분산 환경에서의 사건 처리기의 구조

2.3 분산 처리환경으로 확장시의 고려사항

본 고에서 개발한 실행시간 오류 복구 기법은 분산처리 환경에서도 적용될 수 있다. 멀티 프로세서 시스템으로부터 분산 환경으로 확장되었을 시 그림 3에서의 멀티 프로세서 관리 계층을 변경하여야 한다. 즉, 그림 3에서 보듯이 멀티 프로세서 관리계층 대신 통신부 시스템과 클럭 동기화 계층이 추가되어야 한다.

추가된 계층의 기능은 다음과 같다

(가) 신뢰도 높은 통신부 시스템

분산 처리 환경의 각 노드간의 메세지(message) 전송을 담당하는 계층으로 기존의 네트워크 기능을 이용하나 높은 신뢰도가 요구되며 현재 Prototype에서는 4.3 BSD UNIX의 socket system을 활용하고 있다.

(나) 클럭 동기화 계층

분산 처리 환경하에서 각 노드에서 유지하고 있는 클럭의 동기화가 필요하다. 이때 클럭들간에는 다음과 같은 조건을 충족시키는 것이 필요하다.

조건 1. 내부 근접 동기

같은 Cluster내의 두 노드간의 클럭의 차이는 작은값으로 한계 지워져야 한다.

즉,

$$\max|C_i - C_j| < \delta_{int}$$

여기서 C_i, C_j는 각각 i, j 노드의 클럭값이며 δ_{int}는 하나의 시간 간격(tick)보다 작아야 한다.

조건 2. 생활 시간과의 근접

각 노드의 클럭값은 항상 사용하고 있는 시간값과 가까워야 한다.

즉,

$$|C_i - t| < \delta_{daily}$$

여기서 C_i는 한 노드 i의 클럭값이며 t는 일상 사용하고 있는 시간이다. δ_{daily}는 작을수록 좋다.

조건 3. 외부 근접 동기

Cluster간의 클럭의 차이는 작은값으로 한계 지워져야 한다.

즉,

$$\max|C_{cluster_i} - C_{cluster_j}| < \delta_{ext}$$

조건 4. 클럭 변화율 한계

클럭의 변화율은 클럭이 정확할때는 1이며, 이 변화율은 일정한 값이내로 한계 지워져야 한다.

즉,

$$\left|\frac{dc}{dt} - 1\right| < d_{rate}$$

조건 5. 지역적 인과 관계

같은 노드에서 한 사건 e_1이 다른 사건 e_2 후에 발생하였으면 e_1이 일어난 시간은 e_2가 일어난 시간보다 커야 한다.

즉,

$$T(e_1) > T(e_2)$$

조건 6. 전체적 인과관계

노드 i에서 노드 j로 메세지를 보냈으면 노드 i에서 메세지를 보낸 시간보다 노드 j에서 메세지를 받은 시간이 커야 한다.

즉,

$$T_i\ (\text{message\_send}) < T_j\ (\text{message\_receive})$$

여기서, T_i, T_j는 각각 메세지를 보내고 받았을 때의 노드 i, j의 클럭값을 의미한다.

위의 여섯가지 조건에서 한계치간의 관계는

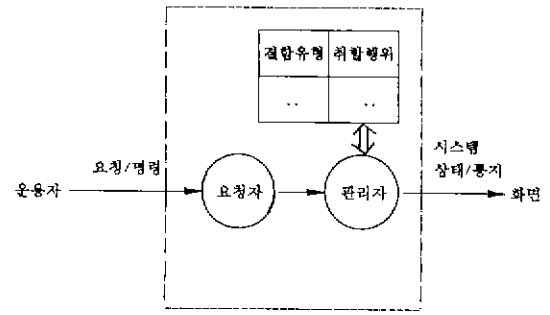

그림 4. 사건 처리기 prototype의 내부 구조

$\delta_{daily} > \delta_{ext} > \delta_{int}$이며 조건 1, 2, 3은 사건이 일어난 시간을 결정하기 위한 조건들이고 조건 4는 두 사건간의 시간 간격을 측정하기 위한 조건이며 조건 5, 6은 사건들이 일어난 순서를 규정하기 위한 조건이다.

이 여섯 조건을 완전히 충족하는 클럭 동기 알고리즘은 아직 알려진 것이 없으나 기존의 네트워크 시스템에서 사용되고 있는 클럭 동기화 알고리즘이 이 조건들을 대부분 충족시키므로 아주 중대한(critical) 작업 이외에는 그대로 응용할 수가 있다.

III. 사건 처리기의 설계 및 구현

이 장에서는 사건 처리기 prototype의 구조 및 기능을 기술한다. 이 Prototype은 사건 처리기의 4계층 중 오류 관리 계층을 구현한 것이고, 멀티프로세서 계층과 시스템 상태 정보 관리 계층은 기존의 SUN 운영체제의 기능을 그대로 활용하였다.

Prototype의 기본 구조는 그림 4에서 보듯이 요청자(requester)와 관리자(manager)로 구성되어 있다.

이 사건 처리기 prototype의 주요기능을 살펴보면 다음과 같다.

- 운용자와 요청자간의 interaction
- 요청자와 관리자간의 interaction
- SUN 운영체제 이용

다음 절들에서 요청자와 관리자의 구조 및 위

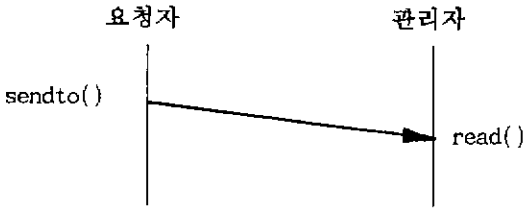

그림 5. 요청자와 관리자의 구성

```
초기화
      ' request_count = 0
      ' request_queue = NULL
      ' request_queue_tail = NULL
interrupt timer 설정
request를 받기 위해 소켓 생성
      ' 읽고자 하는 소켓 생성
      ' 할당된 포트값을 찾아 출력
      ' 리턴 값  descriptor
get status
      ' user와 host의 상태를 얻고 그 이전의 상태를 저장
무한 루프 시작 {
      소켓에서 request를 읽는다.
      request를 패킷에 복사
            ' bcopy( ) 함수를 사용
      packet special_request의 형태에 따라 처리
            ' GET_QUEUE
            ' REMOVE_REQUEST
      새로운 request를 set up
      request를 하나의 화일에 쓰고, yacc를 사용하여
      그 화일을 구문분석
      } 무한 루프 끝
}
```

그림 6. 관리자의 구성

세가지 기능들에 대해 기술한다.

3.1 요청자와 관리자의 구성

요청자와 관리자의 개략적인 구성을 보면 그림 5와 같다.

현 관리자의 구성은 그림 6과 같으며 현재 주기적으로 시스템 상태를 분석하도록 수정중이다.

요청자의 기본구조는 그림 7과 같다.

요청자는 관리자에게 시스템의 상태나, 시스템의 상태에 따른 action을 취해줄 것을 요구하고, 관리자에게서부터 온 request를 받아 적절한 error message나 action을 취하게 된다.

화살표는 요청자가 관리자에게 request를 보내는 것을 의미한다.

```
옵션 또는 request를 받음
      option  : -d : Debbuging을 위한 옵션
                -h : Help 옵션
Request는 file이나 <stdio>로 입력되어 buffer에 저장됨
현재의 운용자 정보를 구하여 저장
     gethostname(rmhost, MAXHOSTNAME);
     getpwuid(getuid());
Request를 관리자에게 보냄
      send_request();
      struct request_packet {
         char host[MAXHOSTNAME];
         int  port;
         char req[MAXREQ];
         char user_id[MAXUSERID];
         int special_request;
         int reqid;
         struct {
               int numrequests;
               REQUEST_LIST request_list;
         } requests
      };  /* request_packet */
관리자의 Port address를 구한다.
Packet을 보내기 위해 buffer로 copy하고 관리자의 값을 설정
      bcopy(&packet, buf, sizeof(packet));
Request를 관리자에게 보내기 위한 send socket을 만듦
      sendto(); 함수 사용
}
```

그림 7. 요청자의 기본 구조

```
create_send_socket(hostname, portnum, nameptr)
char        *hostname;              /* receiver host name      */
u_short     portnum;                /* receiver port number    */
struct      sockaddr_in *nameptr;
{
 socket(AF_INET, sock_DGRAM, 0); /* 보내고자 하는 소켓을 생성  */
                                 /* 생성된 리턴값은 정수형의   */
                                 /* descriptor                 */
 Gethostbyname();                /* 보내고자 하는 소켓의       */
                                 /* 이름을 만든다              */
                                 /* 생성된 리턴값은 host       */
                                 /* 주소를 포함한 하나의 주소  */
}
```

그림 8. 요청자에서 소켓을 생성하는 과정

3.2 요청자와 운용자간의 interaction 및 re-request 처리 과정

요청자는 운용자의 request를 화일이나 keyboard로부터 받는다. 요청자가 운용자에게서 request를 받으면, request 내용(event, action), user_id, hostname, port_number와 timestamp 등을 packet화 한다.

packet의 구조는 다음과 같다

```
create_read_socket(nameptr, msg, portnum)
struct   sockaddr_in *nameptr.
char     *msg.
u_short portnum;
{
socket(AF_INET, sock_DGRAM, 0); /* 읽고자 하는 소켓을 생성  */
                                /* 생성된 리턴값은 정수형의 */
                                /* descriptor               */
     할당된 포트 값을 찾아 출력.
}
```

그림 9. 관리자에서 소켓을 생성하는 과정

event	action	timestamp	user	host	port
deadline	validity-interval	reuse-command	environment		

timestamp field는 request를 보낸 시간이 기록되어 있고, environment field는 실행환경에 필요한 정보가 기록되어 있다.

요청자는 운용자가 요청한 request를 packet화한 후, packet의 user_id와 hosthame, port_number 등을 참조하여 운용자의 request를 관리자에게 보내게 된다.

운용자가 request를 요청자에게 보낼 때에는 간단한 문법을 따르게 된다. 이것은 관리자가 request 내용의 분석을 용이하게 하기 위함이다. 문법은 간단히 event separator action의 형태이다. event는 운용자가 알고자 하는 시스템의 상태정보이고, action은 상태정보에 따라 운용자가 취하고자 하는 행동이다. separator는 기호 "→"을 사용하여, event와 action을 구분하게 한다.

3.3 관리자와 요청자간의 interaction

이 절에서는 관리자와 요청자 사이의 interaction 구조에 대하여 설명한다. 관리자와 요청자의 interaction시에는 socket을 사용한다. 요청자에서 packet화한 운용자의 request는 관리자에서 생성한 port로 socket을 이용하여 관리자에게 보내게 된다. 요청자는 운용자의 request를 관리자에게 전송함으로써 임무를 끝내게 된다.

관리자에서는 request의 내용을 .emreqfile에 쓰고 이 화일을 구문 분석하여 적절한 error message를 출력하거나, request의 action part를 수행하게 된다. 여기서 운용자가 요구한 시스템의 상태정보가 현 시스템의 상태와 맞지 않으면, 운용자의 request를 request queue의 맨 끝에 추가한다.

관리자는 또 주기적으로 현 시스템의 정보를 얻어 저장하는데, 만일 request queue에 있는 request중에 현재 시스템의 상태와 맞는 것이 있으면, 그 request의 action part를 실행한다. 관리자는 시스템에 대한 정보를 MESGFILE (본고에서는, #mesg1)에 기록한다.

3.4 SUN 운영 체제의 응용

관리자는 HOST와 USER의 정보를 갖는다. 호스트의 상태중 whod의 구조체는 /var/spool/whod.* 파일에 있는 정보로서 SUN OS 시스템의 네트워크 환경에서 자동적으로 갱신하여 저장한다. whod.* 파일에는 시스템과 연결된 다른 호스트로부터 주기적으로 시스템의 상태 정보가 입력된다. 이 정보는 whod구조체로 되어 있으며, 이 구조체에 있는 상태변수는 wd_users, wd_type, wd_fill, wd_sendtime, wd_recvtime, wd_hostname, wd_load -av, wd_boottime, struct whoent 등이 있다. 따라서 호스트의 상태는 구조체를 가리키는 포인터를 넘겨 받음으로써 상태를 얻게 된다. hs_nuser는 다른 호스트에서 사용중인 사용자의 수를 말한다. 지금까지 설명한 상태 변수들은 다음과 같은 구조체로 선언되어 있다.

현재 사용하는 사용자의 상태는 /etc/utmp 파일에 저장된 정보를 이용한다. 이 파일에는 사용중인 사용자의 번호(user_id)와 터미널의 번호(tty_id) 그리고 login시간이 저장되어 진다. 이것을 이용하여 user_id, tty_id를 구할 수 있다. idletime은 stat( )라는 SUN OS system call을 이용하여 /dev/tty* 파일의 최근 접근 시간을 구한 다음 현재 시간과의 차이를 구하여 얻게된다. 호스트의 상태는 최근에 입력된 호스트의 상태를 /var/spool/whod.* 파일로부터 받음으로써 구해진다. 여기서 구해진 호스트의 정보를 이용하여 각 호스트의 상태와 사용자의 수를 알 수 있다.

IV. 요약 및 앞으로의 과제

본 고는 멀티 프로세서 상에 실행시간 중에 오류를 감지하고 감지된 오류를 효율적으로 처리할 수 있는 기법을 개발하고 개발된 기법의 효용성을 검증하는 것이다. 현재 독립성, 완결성, 유동성, 확장성 및 강건성 등의 특성을 고려한 실행시간 오류 기법이 개발되었고 이의 효용성 검증을 위해 사건 처리기를 설계하고 그 prototype이 거의 구현 완료 단계에 있다. 앞으로 prototype의 구현을 완료하고 이 prototype을 통해 개발된 기법의 효용성을 검증한 후 필요시 기법을 수정 보완할 것이다.

REFERENCE

1. Abu-Amara, H. H., "Fault-Tolerant Distributed Algorithm for Election in Complete Networks," IEEE Trans. on Computer, Apr. 1988, pp. 449~453.
2. Agrawal, P., "Fault Tolerant in Multiprocessor sysems without Dedicated Redundancy," IEEE Trans. on Conputer, Mar. 1988, pp. 358~362.
3. Avizienis, A. and Kelley, J. P. J., "Fault Tolerance by Design Diversity: Concepts and Experiments," Computer, Vol. 17, No. 8. Aug. 1984, pp. 67~80.
4. Hecht, H., "Fault-Tolerant Software for Real-time Applications," Computing Surveys, Dec. 1976, pp. 391~407.
5. Heu, S., "Experimental Validation of Distributed Recovery Block," '91 Pacific Rim International Symposium on Fault Tolerant Systems, Sep. 1991.
6. Kim, K. H., "Error Detection, Reconfiguration and Recovery in Distributed Processing systems," Proc. IEEE 1st Conf. on Distributed Computing Systems, Oct. 1979, pp. 284~295.
7. Kim, K. H., "Issues in Design of Temporary Blockout Handling Capabilities into Tightly Coupled Computer Networks," Software for Strategic Systems Conference, Oct. 19 88.
8. Randell, B., "System Structure for Software fault tolerance," IEEE Trans. on Software Engr., June 1975, pp. 220~232.
9. Uyar, M.U. and Reeves, A.P., "Dynamic Fault Reconfiguration in a Mesh-Connected MIMD Environment," IEEE Trans. on Computer, Oct. 1988, pp. 1191~1205.
10. 허 신, "멀티프로세서 시스템에서 Runtime 오류 복구에 관한 연구," 최종 연구보고서, 한국전자통신연구소, 1992.

임 기 욱

1977 인하대학교 공과대학 전자공학과 졸업
1987 한양대학교 대학원 전자계산학과 졸업
1990 ~현재 인하대학교 수학과 전자계산전공 박사과정
1977 ~1983 한국전자기술연구소 선임연구원
1983 ~1988 한국전자통신연구소 시스템 소프트웨어 연구실장
1988 ~1989 캘리포니아 주립대학(Irvine) 방문연구원
1989 ~현재 한국전자통신연구소 시스템공학연구부장
관심 분야 : 시스템 소프트웨어, 데이타베이스, 컴퓨터구조