REPORT

(프로세스 심층연구)

과 목 명	운영체제
담 당 교 수	조영석
학 과	컴퓨터정보공학과
학 번	202244012
이 름	노성민
제 출 일	2025.10.20

차례

1. 프로세스의 정의 및 시스템적 관점

프로세스는 1960년대 멀틱스(Multics) 운영체제에서 처음 사용된 개념이므로, 이는 실행 중인 프로그램, 비동기적(asynchronous) 행위, 또는 실행 중인 프로시저의 제어 추적을 의미합니다. 따라서 가장 일반적인 정의는 프로세서에 할당하여 실행할 수 있는 개체, 즉 디스패치(dispatch)가 가능한 대상이라고 할 수 있습니다. 또한, 운영체제에 들어 있는 프로세스 제어 블록(PCB)도 프로세스의 정의 중 하나이므로 프로세스는 운영체제가 관리하는 핵심 단위임을 알 수 있습니다.

시스템 관점에서 프로세스는 실행 순서를 결정하는 역할을 하므로 디스크에 저장된 프로그램에 프로세서를 할당하여 장치나 메모리 같은 파일 자원을 참조합니다. 프로세스는 시스템을 지원하고 협력하여 교착 상태, 보호, 동기화 등의 정보를 교환하므로 시스템 운영의 필수적인 요소입니다.

프로세스와 태스크(스레드)는 구분됩니다. 프로세스는 실행 중인 프로그램이지만, 태스크 또는 스레드(Thread)는 프로세스 내 실행 흐름을 의미합니다. 따라서 프로세스는 메모리가 독립적이고 분리되어 있지만, 스레드는 프로세스 내에서 메모리를 공유합니다. 이로 인해 프로세스 간의 통신(IPC)은 복잡하지만, 스레드 간의 통신은 공유 메모리를 사용하므로 간단하며 동기화가 필요합니다. 또한 프로세스는 생성 비용이 높지만, 스레드는 생성 비용이 낮으므로 경제적인 측면에서도 차이를 보입니다.

2. 프로세스의 제어 및 상태 변화 (4페이지)

운영체제는 프로세서 스케줄러를 이용하여 프로세스의 상태 변화를 관리하므로 프로세스의 생성부터 종료까지의 과정을 수행합니다. 작업 스케줄러는 스풀러가 디스크에 저장한 작업 중 실행할 작업을 선정하여 준비 리스트에 삽입함으로써 다중 프로그래밍의 정도를 결정합니다. 프로세스는 스스로 대기 상태로 가는 것 외에는나머지 상태 변화는 외부 조건으로 발생하므로 운영체제의 역할이 매우 중요합니다.

운영체제는 프로세스 제어 시 필요한 프로세스 상태 정보를 프로세스 제어 블록(PCB, Process Control Block)에 저장하므로 PCB는 특정 프로세스 정보를 저장하는 데이터 블록 또는 레코드(작업 제어 블록 TCB)입니다. 프로세스가 생성되면 메모리에 PCB가 생성되고, 프로세스 실행이 종료하면 해당 PCB도 삭제됩니다. PCB는 프로세스를 관리하려고 유지하는 데이터 구조입니다.

실행 중인 프로세스의 제어를 다른 프로세스에 넘겨 실행 상태가 되도록 하는

것을 문맥 교환(Context Switching)이라고 하므로 문맥 교환이 일어나면 프로세서의 레지스터에 있던 내용을 저장합니다. 문맥 교환은 인터럽트가 발생하거나 프로세스가 '준비 → 실행', '실행 → 준비', 또는 '실행 → 대기' 상태로 바뀔 때 발생합니다. 문맥 교환은 이전 프로세스의 상태 레지스터 내용을 보관하고 다른 프로세스의 레지스터를 적재하여 프로세스를 교환하는 일련의 과정이므로 이 과정에서 오버헤드가 발생합니다. 오버헤드는 시간 비용을 소요하므로 운영체제 설계 시 불필요한 문맥 교환 감소가 주요 목표가 됩니다.

3. 프로세스의 생성과 종료

프로세스는 실행 중 프로세스 생성 시스템 호출을 이용하여 새로운 프로세스를 생성할 수 있으므로 부모 프로세스(parent process)와 자식 프로세스(child process) 관계를 유지하여 계층적 생성을 이룹니다. 프로세스 생성 순서를 저장하고 부모-자식 관계를 유지하여 계층 구조를 형성합니다.

새로운 프로세스 생성 시 여러 세부 작업 순서가 필요합니다. 첫째, 새로운 프로세스에 프로세스 식별자를 할당하며, 둘째, 프로세스의 모든 구성 요소를 포함할 수 있는 주소 공간과 PCB 공간을 할당합니다. 셋째, 프로세스 상태, 프로그램 카운터 등을 포함하는 PCB를 초기화하며, 넷째, 해당 큐에 삽입하는 링크 작업을 수행하므로 프로세스 생성은 복잡한 과정을 거칩니다.

프로세스 생성 후 부모 프로세스와 자식 프로세스는 동시 실행되거나 부모 프로세스는 자식 프로세스 모두 종료할 때까지 대기할 수 있습니다. 참고로, C++ 표준 라이브러리는 프로세스를 직접 생성하는 기능을 제공하지 않으므로 POSIX(Linux, macOS) 환경에서는 fork()와 exec()를, Windows 환경에서는 CreateProcess()API를 사용해야 합니다.

프로세스의 종료는 프로세스가 마지막 명령 실행 후 운영체제에 삭제를 요청하거나 abort()명령어로 프로세스가 종료될 수 있으므로 정상 종료 외에도 시간 초과, 산술 오류, 보호 오류, 메모리 부족 등의 다양한 이유로 종료됩니다. 부모 프로세스가 종료하면 운영체제가 자식 프로세스를 연속 종료시키는 것이 보통이므로 부모 프로세스는 자식 프로세스가 할당된 자원을 초과하여 사용할 때 자식 프로세스를 종료시킬 수도 있습니다. 프로세스 제거 시에는 사용하던 자원을 시스템에 돌려주고, 해당 프로세스는 시스템 리스트에서 사라져 PCB를 회수하게 되므로 프로세스를 파괴하는 과정이 발생합니다.

4. 스레드의 개념 및 이점

스레드(thread)는 프로세스의 특성인 자원과 제어 중에서 제어만을 분리한 실행 단위이므로 프로세스 하나는 스레드 한 개 이상으로 나눌 수 있습니다. 스레드는 프로세스의 직접 실행 정보를 제외한 나머지 프로세스 관리 정보를 공유하므로 같은 프로세스의 스레드들은 동일한 주소 공간을 공유합니다. 이로 인해 스레드는 경량 프로세스(LWP, Light Weight Process)라고 불리기도 합니다.

스레드는 프로세스보다 문맥 교환이 훨씬 경제적입니다. 스레드별로 실행 환경 정보가 따로 있지만 서로 많이 공유하므로 동일한 프로세스의 스레드에 프로세서를 할당하는 것이 효율적이기 때문입니다. 또한 스레드 생성은 프로세스 생성 및 종료보다 오버헤드가 훨씬 적으므로 경제성이 좋다고 평가됩니다.

스레드 병렬 수행의 주요 이점은 다음과 같습니다. 첫째, 사용자 응답성이 증가합니다. 둘째, 프로세스의 자원과 메모리를 공유할 수 있습니다. 셋째, 다중 처리(멀티프로세싱)를 통해 성능과 효율이 향상되므로 작업량을 증가시키고 시스템 전체의 성능을 향상시킵니다. 다중 스레드 시스템에서는 서버 스레드 한 개가 대기 상태일 때 동일 프로세스의 다른 스레드가 실행 가능하므로 프로세서가 하나인 시스템에서 발생하는 대기 상태 문제를 해결할 수 있습니다.

스레드 한 개가 대기 상태로 변환되더라도 전체 프로세스가 대기 상태로 변환되지 않으므로 실행 상태의 스레드가 대기 상태가 되면 다른 스레드가 실행 가능합니다. 하지만 동일한 프로세스에 있는 전체 스레드는 프로세스의 모든 주소에 접근 가능하여 스레드 한 개가 다른 스레드의 스택 읽기나 덮어쓰기가 가능하므로 스레드 간에 보호는 하지 않습니다. 스레드 제어 시에는 스레드 제어 블록(TCB, Thread Control Block)에 정보를 저장하여 관리합니다.

5. 스레드의 운영체제 구현 유형

스레드는 운영체제의 지원 방식에 따라 세 가지 유형으로 구현되므로각 방식의 장단점을 이해하는 것이 중요합니다.

5.1 사용자 수준 스레드 (User-Level Thread)

사용자 수준 스레드는 다대일(n:1) 매핑 방식을 사용하며, 사용자 영역의 스레드 라이브러리로 구현됩니다. 스레드 교환에 커널이 개입하지 않아 커널에서 사용자 영역으로 전환이 불필요하므로 오버헤드가 적습니다. 또한 커널에 독립적 스케줄링이 가능하므로 모든 운영체제에 적용할 수 있는 높은 이식성을 가집니

다. 하지만 커널은 스레드가 아닌 프로세스를 한 단위로 인식하므로 동일한 프로세스의 스레드 한 개가 대기 상태가 되면 이 중 어떤 스레드도 실행 불가하여 시스템의 동시성을 지원하지 못하는 단점이 있습니다.

5.2 커널 수준 스레드 (Kernel-Level Thread)

커널 수준 스레드는 일대일(1:1) 매핑 방식을 사용하며, 커널이 스레드와 관련된 모든 작업을 관리합니다. 커널이 직접 스케줄링하고 실행하기 때문에 스레드 한 개가 대기 상태가 되어도 동일한 프로세스에 속한 다른 스레드로 교환 가능합니다. 이는 사용자 수준 스레드의 한계를 극복하는 방법이므로 다수의 스레드가 프로세서 할당을 받아 병행 수행할 수 있습니다. 그러나 커널이 전체 프로세스와 스레드 정보를 유지하여 오버헤드가 커지며 스케줄링과 동기화를 하려면 더 많은 자원이 필요합니다. 윈도우 NT·XP·2000, 리눅스 등이 대표적입니다.

5.3 혼합형 스레드 (Multiplexed Thread)

혼합형 스레드는 사용자 수준 스레드와 커널 수준 스레드를 혼합한 다대다(n:m) 매핑 구조이므로 앞선 두 유형의 문제점을 극복하려 합니다. 이 방식에서는 스레드 라이브러리가 최적의 성능을 지원하도록 커널이 경량 프로세스 수를 동적으로 조절하며 스레드 풀링(thread pooling)을 이용하여 오버헤드를 감소시킬 수 있습니다. 스레드 풀링은 미리 생성한 스레드의 재사용으로 스레드 생성 시간을 줄여 시스템의 부담을 경감시키므로 시스템 자원 소비 감소와 전체 성능 유지를 가능하게 합니다.

6. 레포트 작성에서 느낀점

이번 심층연구 과정에서는 운영체제에 관한 교재와 참고 자료를 여러 번 반복해 읽으면서 이론적 내용을 깊이 있게 이해하는 데 집중했습니다. 교재를 두 번, 세 번씩 정독하며 핵심 개념과 구조, 그리고 여러 용어들의 의미를 체계적으로 정리할 수 있었고, 반복적인 학습이 이해도 향상에 매우 큰 도움이 되었습니다.

특히 프로세스와 스레드 구조, PCB와 TCB 등 복잡한 시스템 구성 요소들을 꼼꼼히 읽고 정리하면서 표면적인 암기 이상의 깊은 통찰을 얻을 수 있었고, 앞으로 관련 분야를 공부하거나 개발할 때 교재를 반복해서 읽는 노력이 큰 자산이 될 것임을 느꼈습니다.

이번 심층연구에서는 실습 없이 교재와 참고 자료를 여러 번 꼼꼼히 읽으면서

운영체제의 이론적 개념을 깊이있게 정리하는 데 집중했습니다. 교재를 두 번, 세 번씩 정독하며 복잡했던 프로세스 구조와 다양한 용어들을 체계적으로 이해할 수 있었고, 반복 학습의 중요성을 실감할 수 있었습니다.

특히 PCB, TCB 등 운영체제의 핵심 구조와 각 요소의 역할에 대해 교재 내용을 중심으로 깊이 있게 파악하면서 단순 암기에 그치지 않고 실제 시스템 동작 원리를 체득할 수 있었습니다. 이번 과정은 앞으로 운영체제 분야나 시스템 개발을 공부할 때에도 교재 반복 학습과 꼼꼼한 내용 정리가 매우 유익하다는 점을 느끼게 해주었습니다.