Multithread

보충자료

목포해양대 해양전자통신공학부

병렬처리 실현 방안

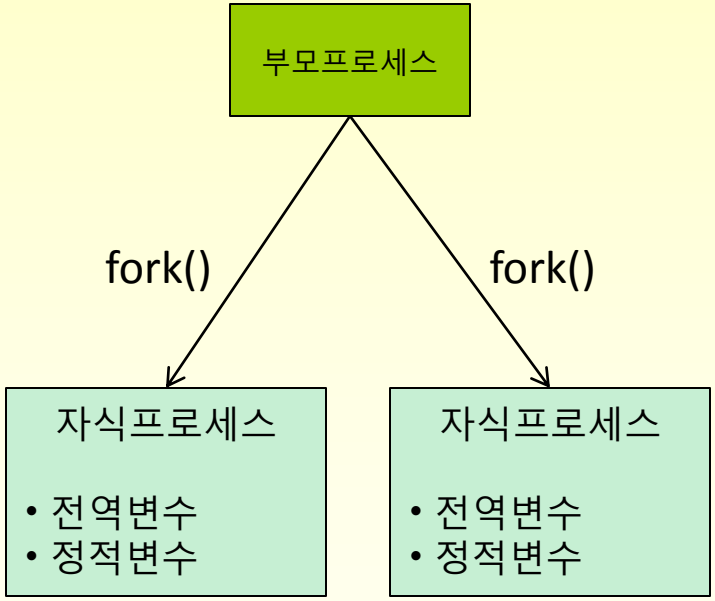
- 프로그램 자체를 병렬처리가 가능하도록 작성
  - 짧은 시간에 끝나는 단위로 처리를 분할
  - 어디까지 수행했는지 관리하면서 타이머 등에 의해 나누어 처리
  - 병렬 프로세서/병렬 프로그램
- 미리 여러 개의 프로세스 실행시켜 일을 분담
- fork( )를 이용하여 하위 프로세스를 실행
- 멀티 쓰레드 사용

다중 프로세스와 멀티 쓰레드(1)

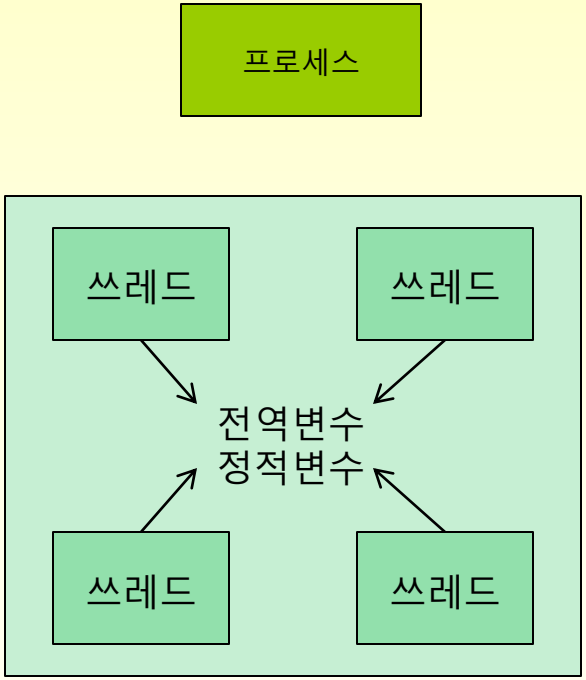
- 다중 프로세스
  - 프로세스 공간이 독립적이므로 전역변수나 정적(static) 변수 사용에 어려움이 없음
  - 개별적으로 디버깅 용이
  - 하나의 프로세스에 대한 제한(동시에 열 수 있는 파일 수 등)에 덜 민감
  - 자식 프로세스의 종료가 전체에 영향을 미치지 않음
  - 자식 프로세스 처리에 많은 메모리가 필요해도 종료되면 모든 메모리가 반환됨
  - 프로세스 공간이 독립적으로 필요하므로 메모리 소비량이 상대적으로 많음
  - 프로세스 자체가 큰 경우 fork() 수행 시 시간 소요
  - 유닉스 계열이 아닌 OS에서는 지원이 안되는 경우도 있음
  - 프로세스 동기 등을 위한 IPC(프로세스 간 통신) 필요
  - wait()에 대한 관리가 부실한 경우 좀비(zombie) 프로세스 잔존
- 멀티 쓰레드
  - 하나의 프로세스 공간 만으로 병렬 처리를 하므로 메모리 공간 절약
  - 쓰레드 생성 시 소요되는 시간이 적음
  - 대부분의 OS에서 제공하는 추세
  - 쓰레드 종료 상태를 wait() 할 필요가 없음
  - 배타 처리 등이 제공됨
  - 전역변수나 정적 변수 사용에 어려움
  - 디버깅 수행이 어려움
  - 하나의 프로세스에 대한 제한(동시에 열 수 있는 파일 수 등)으로 인해 쓰레드 개수가 한정됨
  - 쓰레드 도중 종료가 어려움
  - 쓰레드 내에서 동적으로 메모리를 확보하는 경우 전체 프로세스 크기가 커짐

다중 프로세스와 멀티 쓰레드(2)

- 다중 프로세스

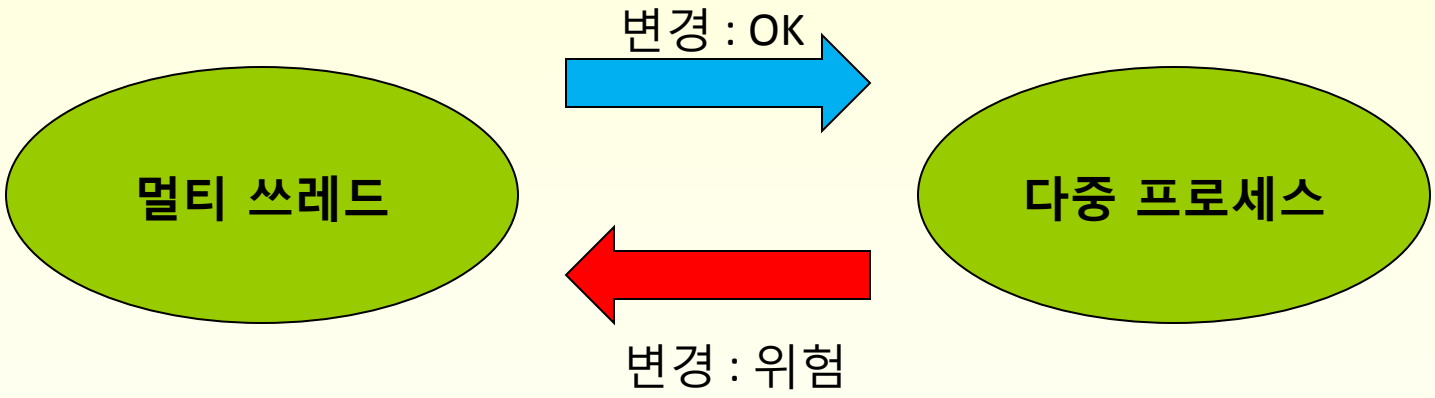


- 멀티쓰레드



다중 프로세스와 멀티 쓰레드(3)

- 멀티 쓰레드로 작성된 프로그램을 다중 프로세스로 변경하는 것은 용이
- 다중 프로세스로 작성된 프로그램을 멀티쓰레드로 변경하는 것은 위험
  - 멀티 쓰레드의 경우 따라야 하는 제약이 많기 때문
  - 전역 변수, 정적 변수의 취급이 특히 어려움



다중 프로세스 예제 (1)

```
#include	<stdio.h>
#include	<sys/types.h>
#include	<sys/wait.h>

void counter()
{
		int	i;
		pid_t	pid;

		pid=getpid();

		for(i=0;i<10;i++) {
				sleep(1);
				printf("[%d]%d\n",pid,i);
		}

		return;
}
```

다중 프로세스 예제 (2)

```
void main()
{
        pid_t   p_pid,pid;

        p_pid=getpid();

        printf("[%d]start\n",p_pid);

        switch(pid=fork()){
                case    0:      /* child */
                                counter();
                                exit(0);
                case    -1:
                                perror("fork");
                                break;
                default:       /* parent */
                                printf("[%d]child pid = %d\n",p_pid,pid);
                                break;
        }
```

다중 프로세스 예제 (3)

```
        switch(pid=fork()){
                case  0:    /* child */
                            counter();
                            exit(0);
                case  -1:
                            perror("fork");
                            break;
                default:    /* parent */
                            printf("[%d]child pid = %d\n",p_pid,pid);
                            break;
        }

        pid=wait(NULL);
        printf("[%d]pid = %d end\n",p_pid,pid);
        pid=wait(NULL);
        printf("[%d]pid = %d end\n",p_pid,pid);

        printf("[%d]end\n",p_pid);
}
```

멀티쓰레드 예제 (1)

```
#include      <stdio.h>
#include      <sys/types.h>
#include      <pthread.h>

void *counter(void *arg)
{
              int    i;
              pid_t   pid;
              pthread_t    thread_id;

              pid=getpid();
              thread_id=pthread_self();

              for(i=0;i<10;i++){
                            sleep(1);
                            printf("[%d][%d]%d\n",pid,thread_id,i);
              }

              return(arg);
}
```

멀티쓰레드 예제 (2)

```
void main()
{
        pid_t   p_pid;
        pthread_t    thread_id1,thread_id2;
        int          status;
        void         *result;

        p_pid=getpid();

        printf("[%d]start\n",p_pid);
        status=pthread_create(&thread_id1,NULL,counter,(void *)NULL);
        if(status!=0){
                fprintf(stderr,"pthread_create : %s",strerror(status));
        }
        else{
                printf("[%d]thread_id1=%d\n",p_pid,thread_id1);
        }
```

멀티쓰레드 예제 (3)

```
        status=pthread_create(&thread_id2,NULL,counter,(void *)NULL);
        if(status!=0){
                fprintf(stderr,"pthread_create : %s",strerror(status));
        }
        else{
                printf("[%d]thread_id2=%d\n",p_pid,thread_id2);
        }

        pthread_join(thread_id1,&result);
        printf("[%d]thread_id1 = %d end\n",p_pid,thread_id1);
        pthread_join(thread_id2,&result);
        printf("[%d]thread_id2 = %d end\n",p_pid,thread_id2);

        printf("[%d]end\n",p_pid);
}
```