韓國心理學會誌 : 生物 및 生理
Korean Journal of Biological and Physiological Psychology
1991. Vol. 3, 156-161

《자 료》

개인용 컴퓨터를 이용한 인터럽트 방식의 시간통제

이 두 현

고려대학교 심리학과

최근 각종 심리학 실험에 개인용 컴퓨터를 많이 이용하고 있다. 이러한 컴퓨터를 이용한 실험에는 통상적으로 시간통제가 필요한 바, 반응시간 측정, 자극제시통제등에 정확한 시간통제가 요구된다. 개인용 컴퓨터에서 시간을 통제하는 방법에는 여러가지 방식이 있을 수 있으나 본 연구에서는 컴퓨터가 다른 작업을 하면서 동시에 시간을 정확히 통제, 계산할 수 있는 인터럽트 방식의 프로그램을 작성하였다.

개인용 컴퓨터가 보급되기 시작한 이후 심리학에서도 컴퓨터를 이용한 실험이 증가하고 있다. 이러한 실험들에서 컴퓨터가 하는 역할은 다양하다. 즉, 컴퓨터 화면에 문자, 도형과 같은 자극을 일정한 시간 계획하에 제시하거나, 자극제시후 피험자의 반응 시간을 측정하는데 이용된다. 또한 동물 실험의 경우, 피험동물에게 여러가지 자극을 제시하기 위한 실험기구들의 통제 내지는 동물의 반응을 측정하는데 이용된다(이두현·김현택·류재욱·김기석, 1990). 이 모든 경우에 있어서 가장 중요한 컴퓨터의 역할은 정확한 시간통제에 있다.

따라서 실험을 수행하기 위한 컴퓨터 프로그램을 개발하는데 있어서 시간통제 프로그램은 가장 중요한 비중을 차지하고 있다. 이러한 시간통제 프로그램을 작성하는 방법에는 크게 두가지 방법이 있을 수 있다. 첫째는 프로그램 방식으로서 일정한 횟수동안 명령어를 반복시켜 그 명령어의 수행시간과 반복횟수를 곱하여 시간계산을 하는 방법이 있는데 이는 과거 Apple II 컴퓨터에서 많이 이용되었다(강은주, 1986; 이두현, 1986). 그러나 이러한 방법은 여러가지 단점을 지니고 있는데, 첫째는 계산된 시간이 실제의 시간 경과와 차이가 많고 두번째는 시간통제 프로그램이 수행되고 있는 동안은 다른 작업을 하지 못한다는 것이다. 따라서 실험의 융통성이 줄어든다. 두번째 시간통제 방법으로는 하드웨어 인터럽트 방식으로 컴퓨터 외부에서 미리 정해진 시간간격으로 컴퓨터의 중앙처리장치(CPU)에 인터럽트 요청을 하도록 인터럽트 카드를 설계하여 컴퓨터로 하여금 다른 작업을 수행하고 있다가 외부 인터럽트 요청이 있을때만 잠깐 시간 계산을 하고는 다시 하던 작업을 계속 수행하게끔 하는 방식이다. 이는 컴퓨터의 중앙처리장치에 하드웨어 인터럽트 요청 단자가 있음으로서 가능하다.

이에 Apple II 컴퓨터에서는 타이머 인터럽트 카드를 설계·제작하여 컴퓨터와 인터페이스를 시켜야 하드웨어 인터럽트 방식이 가능하다. 그러

나 최근 널리 보급되어 있는 IBM 개인용 컴퓨터의 경우 이러한 하드웨어 타이머 인터럽트 장치가 내장되어 있어, 시스템의 시간·날짜 계산 및 스피커로의 소리 출력에 이용되고 있다(한국성, 1989; Norton, 1985). 그러나 이 내장된 인터럽트 장치는 약 55msec마다 인터럽트 요청이 있도록 설계되어 있어, 경우에 따라서는 1msec 단위까지 통제해야 하는 실험의 경우는 사용하기가 어렵다.

따라서 본 연구의 목적은 IBM 개인용 컴퓨터의 하드웨어 타이머 인터럽트 장치를 소프트웨어적으로 수정하여 실험자가 원하는 시간간격으로 인터럽트를 요청할 수 있는 프로그램을 작성하는데 있다.

방 법

본 연구의 타이머 인터럽트 프로그램은 TURBO C 2.0을 사용하여 작성하였다. IBM 개인용 컴퓨터의 하드웨어 타이머 인터럽트 장치는 자체 클락(clock)을 이용하는데, 이 클락은 1.19318㎒로 발진을 한다. 하드웨어 타이머 인터럽트 장치는 클락의 발진횟수를 셈하다가 그 발진횟수가 포트 64번지에 저장되어 있는 값과 일치할 때 하드웨어 인터럽트를 발생시키고 다시 클락의 발진횟수를 셈하는 작업을 반복한다. 통상적으로 포트 64번지에는 65,536이라는 수가 저장되어 있는데, 이 때문에 클락횟수가 매 65,536번째에 도달했을때 인터럽트를 발생시킨다. 따라서 1초동안의 클락발진 횟수인 1,193,180을 65,536으로 나눈 값이 1초에 인터럽트가 발생한 횟수이며 그 값은 약 18.2회가 되고 인터럽트 발생 간격은 약 55msec가 된다. 그러나 IBM 개인용 컴퓨터는 포트 64번지에 있는 65,536이라는 숫자를 변경 시킬 수 있는 방법을 제공하는데, 이때문에 인터럽트 발생 간격을 조정할 수 있다. 즉, 예를 들어 1초에 1,000번, 또는 1msec 간격으로 인터럽트를 발생시키려면 포트 64번지에 1,193,180을 1,000으로 나눈 숫자인 1,193을 입력시키면 된다. 그러면 1초에 발생되는 인터럽트 횟수는 1,193,80을 1,193으로 나눈 값, 즉, 1,000.15088회가 되고, 이는 1.000150msec의 시간간격이 된다. 이는 1msec당 150 nanosec(1nanosec=1sec/1,000,000,000)의 오차를 갖게 된다. 이를 초당 오차로 계산하면 150 nanosec × 1000은 150 microsec가 되고, 분당 오차로 계산하면 150 microsec × 60은 9msec가 된다. 이를 다시 시간당 오차로 계산하면 9msec × 60은 540msec의 오차가 된다. 따라서 한 시간에 약 0.5초의 오차가 발생하는 시간통제를 할 수 있는 것이다. 이런식으로 오차계산까지 정확하게 할 수 있기 때문에 실제의 시간을 계산할때 보정이 가능하다. 표 1은 포트 64번지에 실험자가 원하는 인터럽트 발생 클락발진 횟수를 입력시키는 프로그램이다.

프로그램에서 첫번째 함수인 void int-time (int freq)은 사용자가 원하는 초당 인터럽트 발생 횟수를 계산하여 그에 필요한 클락 발진횟수를 포트 64번지에 입력하는 프로그램이다. 따라서 초당1,000회의 인터럽트를 발생시키고자 할 경우 다음과 같은 프로그램을 작성하면 된다.

```
int frequency=1000;
int-time(frequency);
```

이상의 프로그램이 실행이 되면 클락이 65,536회 발진할때마다 발생되던 인터럽트가 1,193회 발진할때마다 발생된다. 포트 64번지에 필요한 값을 입력한 후에는 하드웨어 인터럽트가 발생했을 경우 수행하여야 할 프로그램을 작성하여야 한다. 정상적인 경우의 IBM 개인용 컴퓨터는 이때 인터럽트 발생 횟수를 셈하여 일정한 메모리에 그 결과를 저장하였다가 후에 시스템의 시간 및 날짜를 계산하는데 사용한다. 이때 필요로 되는 프로그램은 메모리 32, 33, 34, 36번지에 저장된 값을 어드레스로 하는 메모리에 위치하게 된다. 그러나 어떤 실험을 위한 프로그램을 작성할 경우 인터럽트 발생시 그 실험에 고유한 프로그램을 필요로 한다. 예를 들면 인터럽트가 발생할 때마다 경과된 시간을 계산하거나, A/D변환의 변환작업을 수행할 수 있다. 이러한 프로그램 함수는 세번째 함수인 void interrupt cal-time ()에서 수행된다. 즉, 하드웨어 인터럽트가 발생

표 1. TIME.C 프로그램. 하드웨어 인터럽트 발생횟수를 변경시키는 프로그램

```
#include <dos.h>
void int_time(int ctime);
void interrupt cal_time();
void install(void interrupt(*faddr)(),int inum);
void restore();
long acc;  /* acc==> 인터럽트가 걸린 횟수 */
int x,y;   /* x,y==> 인터럽트 8번의 어드레스를 저장할 변수 */

void int_time(int freq)
{
 int status=54,ti=8;
 long clock=1193180,count;
    short hcount,lcount;
    count=clock/freq;
    hcount=count/256;
    lcount=count-(hcount*256);
    outportb(67,status);
   /* 타이머 명령어 레지스터 포트 67에 00110110b를 출력 */
    outportb(64,lcount);
    outportb(64,hcount);
   /* 타이머 0에 계산된 counter수를 출력  */
    x=peek(0,8*4);
    y=peek(0,8*4+2);
   /* 인터럽트 8의 어드레스를 저장하고 */
    poke(0,99*4,x);
    poke(0,99*4+2,y);
   /* 그 값을 인터럽트 99로 옮김.  따라서 INT8이 INT99로 바뀜 */
    install(cal_time,ti);  /* INT8에 시간계산 프로그램을 저장 */
}

void restore()      /* 초기의 상태로 환원  */
{
        short lcount=255,hcount=255,status=255;
        outportb(67,status);
        outportb(64,lcount);
        outportb(64,hcount);
        poke(0,8*4,x);
        poke(0,8*4+2,y);
}

void interrupt cal_time()     /* 새로 설정된 TIMER에 의한 시간계산 프로그램 */
{
        int tri=99;
        acc++;                  /* 인터럽트가 걸릴때마다 1씩 증가 */
        geninterrupt(tri);      /* 원래의 INT8을 수행  */
```

```
}

void install(void interrupt(*faddr)(),int inum)
/* 시간계산 프로그램을 INT8로 변환 */
{
        disable();
        setvect(inum,faddr);
        enable();
}
```

표 2. TIME.C를 이용하는 예제 프로그램. 스페이스 바를 누를때마다 그 시간간격을 출력하는 프로그램

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "time.c"
int SPACE=32;
main()
{
   int key,frequency=1000;  /* frequency가 1000이면 1msec마다 인터럽트 발생 */
   long ti;  /* 시간 계산용 변수 */
   char imsi[10];
        int_time(frequency); /* TIMER설정 */
        clrscr();
        gotoxy(10,1);
        printf("Timer control example by D.H.Lee");
        gotoxy(10,23);
        printf(".....To stop press the 'ESC' key!");
        gotoxy(25,12);
        acc=0;    /* 시간의 초기화 */
     do {   key=getkey();
        if (key==SPACE) { ti=acc;  /* 스페이스 키가 눌렸으면 acc를 읽고 */
        acc=0;                     /* 다시 초기화 */
        ti=ti*(1000.0/frequency);  /* msec단위로 계산 */
        ltoa (ti,imsi,10);         /* 화면에 출력 */
        gotoxy(25,12);
        printf("                    ");
        gotoxy(25,12);
        printf("%s millisecond",imsi);
```

```
        }
       }while(key!=27);
        restore();
}

int getkey()            /* 눌러진 키의 아스키 값 계산 */
{
       int key,lo,hi;
       if(kbhit()!=0) {
              key=bioskey(0);
              lo=key&0x00ff;
              hi=(key&0xff00)>>8;
              return((lo==0) ? hi+256:lo);
              }
        return(0);
}
```

하면 중앙처리장치는 하던 작업을 잠시 중단하고 void interrupt cal-time()을 수행한 후 다시 하던 작업을 계속한다. 따라서 본 연구에서는 void interrupt cal-time() 함수에서 인터럽트가 발생된 횟수를 계산하도록 프로그램을 구성하였는데 실험 목적에 따라 이 부분은 변경될 수 있다. 예를 들어 이 부분에 A/D변환 프로그램을 작성해 놓으면 정해진 시간간격마다 A/D변환 작업이 자동적으로 일어난다. 두번째 함수인 void restore()는 더 이상 하드웨어 인터럽트 발생이 필요치 않을 경우 초기 상태로 환원시키는 프로그램이다.

표 2는 실제로 표 1의 함수들을 이용하는 예제 프로그램으로서 스페이스 바를 누를 때마다 그 시간간격을 msec 단위로 출력시켜 주는 프로그램이다. main()함수에서 인터럽트를 1msec마다 발생시키도록 frequency값을 1,000으로 결정하였다.

결 과

이러한 프로그램을 여러 실험에 응용한 결과 정확한 시간통제 효과를 얻을 수 있었다(이두현 등, 1990; 이두현, 1991) 그러나 몇가지 문제점도 발견되었다. 첫번째 문제는 하드웨어 인터럽트 프로그램이 수행되고 있는 동안은 보조기억장치(하드 디스크 및 플로피 디스크)에 읽고 쓰기가 불가능 하였다. 두번째 문제는 하드웨어 인터럽트 프로그램이 수행되고 있는 동안, 또는 수행이 된 후에 시스템의 시간이 변경되었다. 이는 시스템에서 1초에 18.2회씩 인터럽트가 발생하는 것으로 계산을 하기 때문이었다. 따라서 시스템의 시간을 이용하는 작업이 불가능 하였다.

논 의

이상의 몇가지 문제점이 존재하나 본 프로그램은 여러가지 심리학 실험에 응용될 수 있다. 그리고 시스템의 시간이 변하는 것은 TIME.C의 void interrupt cal-time() 함수에 시스템의 시간을 보정해 주는 프로그램을 보충해 줌으로서 해결할 수 있을 것이다. 본 프로그램의 응용시 한가지 주의할 점은 void interrupt cal-time() 함수에서 실제 경과시간을 나타내주는 변수 acc

의 초기화에 있다. 즉, 실제로 실험 프로그램에서는 시간 측정을 시작하는 시점에서 초기화 시켜야 한다. 그러나 프로그램 작성시 이것이 여의치 않을 경우도 있다.

참고문헌

강은주(1985). 해마손상이 지연 과제 수행에 미치는 영향. 고려대학교 석사학위 청구논문.

이두현(1985). 순막조건반응에서 맥락변화가 잠재적억제에 미치는 효과. 고려대학교 석사학위 청구논문.

이두현(1991). 내측중격의 손상과 자극이 해마 θ 파, 해마 다단위활동 및 고전적 순막조건반응의 소거에 미치는 효과. 고려대학교 박사학위 청구논문.

이두현 · 김현택 · 류재욱 · 김기석(1990). 고전적 순막 조건화동안의 해마 뇌전도와 다단위 활동. **한국심리학회지 : 생물 및 생리**, 2, 69-78.

한성국(1989). 시스템 프로그래머를 위한 IBM PC XT/AT 기술사전. 집문당.

Norton, P.(1985). Programmer's guide to the IBM PC. Microsoft Press.

韓國心理學會誌 : 生物 및 生理

Korean Journal of Biological and Physiological Psychology

1991. Vol. 3, 156-161

Time control using interrupt on IBM PC

Doo-Hyun Lee

Korea University

Recently, personal computer in generally used for various psychological experiments. These experiments usually are needed to control time with unit of millisecond in measuring response time, controlling stimulus presentation, etc. There are a number of means to control time in personal computer, this study show the interrupt method that experimenters con control and measure real time in IBM personal computer without preventing current user's works. This method was programmed with C language using Turbo C 2.0 compiler.