한국부식학회지
J. Corros Sci. Soc. of Korea
Vol. 25 No. 1, March, 1996
〈연구논문〉

# PC를 이용한 멀티테스킹 데이터수집

연 제 원 · 이 영 욱* · 최 인 규
한국원자력연구소, 화학특성규명기술개발분야, *핵연료개발그룹

# Multi-tasking Data Aquisition Using A Personal Computer

Jei-Won Yeon, Young Ouk Lee*, In Kyu Choi
*Korea Atomic Energy Research Institute*
*Radioactive Waste Chemistry Research Development,*
**Nuclear Fuel Development Group*

**The advantages of multi-tasking and network operations applied to long term corrosion experiments are described. These can be achieved by using an operating system with Unix functions in the IBM PC. With the client/server concept, data aquisition modules are programmed to control the analog-to-digital converter devices.**

## 1. 서 론

부식연구에서 전기화학적 측정방법은 부식속도를 신속하게 측정하고 부식현상을 규명하기 위하여 많이 사용된다. 현재 자체 중앙처리장치(CPU)를 가지는 많은 종류의 우수한 다기능 전기화학 부식측정기들이 상품으로 개발되어 있다. 그러나 전위 및 분극저항의 장기 모니터링,[1-3] 전기화학적 wet/dry 시험[4]등과 같이 측정에 오랜 시간이 필요한 경우에 이러한 장비의 사용은 효율성 측면에서 적합하지 않으므로 일반적으로 실험실에서 간단한 아날로그 전위측정기나 자체 개발한 앰프 등을 ADC(analog-to-digital converter)가 내.외장된 PC(personal computer) 에 연결하여 데이터를 수집 관리하여 왔으며,[5] 현재도 많은 부식실험에서 이러한 장치를 사용하고 있다.[6,7] 이를 위한 프로그램은 싱글테스킹 운영체제에서는, 여러 채널을 사용할 수 있는 ADC에서도 한 채널의 사용도중 비어 있는 다른 채널의 추가 사용은 복잡한 프로그래밍 작업을 필요로 한다. 그리고 가속화 대기부식측정에 사용되는 전기화학적 wet/dry 시험과 같이 모터, 릴레이제어, 부식전위, 온도 및 습도 등을 모니터링하는 시스템의 경우 제어 및 측정에 대한 모든 기능을 하나의 프로그램으로 구현하여야 하므로 프로그래밍 작업이 복잡해지는 단점을 가지고 있다.

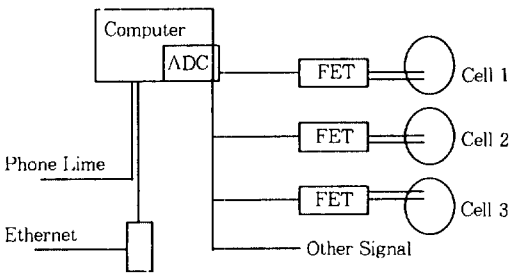
IBM 호환기종 386이상 PC에서 사용할 수 있는 유닉스 계열의 시분할 운영체제가 개발되면서 저가의 PC로도 멀티테스킹(multi-tasking)과 서버컴퓨터로의 사용이 가능한 환경을 만들 수 있게 되었으나[8-10] 데이터 수집에 필수적인 PC용 디바이스인 AD/DAC(analog-to-digital/digital-to-analog converter) 제조회사들이 아직 이

러한 운영체제를 위한 디바이스 드라이버를 공급하지 않으므로 이 환경에서 데이터 수집을 실험에 적용한 예는 찾아보기 어렵다.

이 글에서는 유닉스 기능을 가지는 IBM 호환 PC(386 이상)용 운영체제(Linux)에서 대부분의 AD/DAC에 적용할 수 있는 데이터 수집 저장 프로그래밍 방법을 보이고 이러한 기법을 사용함으로서 얻는 편리함을 간단한 실험결과와 함께 기술하였다.

## 2. 시스템구성 및 프로그램

측정시스템의 구성은 운영체제와 관계없이 동일하다. Fig. 1은 원자력연구소 화학특성규명기술개발분야 실험실에서 사용하는 시스템의 세부사양인데 전기화학적 부식쎌의 부식전위와 다른 계측기에서 나오는 신호를 측정할 수 있는 기능을 가진다.




Computer : IBM compatible 486 DX2 66
O/S : Slackware, Linux 1.1.90
FET : Potentiometer
Cell : Corrosion Cell
ADC : Analog to Digital Converter, PCL 711s, 12bit

Fig. 1. Corrosion Potential Monitoring System.

```
#include <asm/io.h>
#include <asm/unistd.h>
#include <stdio.h>
#include <time.h>

time_t *a;
main()
  {
    FILE *fp;
    char ch;
    char data[10];
    int port, i, j, dh, dl, dta, dta_total, ii, chn, t0.
    diff=0, st_aq, end_aq,tz;
    float v, dta_mean, volt;
    port= 0x220;
    printf("Enter Channel port number(0-7):");
    scanf("%d", &chn);
    while(getchar()!= '\n');

    printf("Enter Data file name:");
    scanf("%s", &data);
    printf("%s\n", data);

    t0=tz=time(a);
for (i=1 ;i<600 ;i++) {
    do { sleep(1);
        diff=time(a)-t0; st_aq=time(a)-tz; }
    while(diff < 100);
    ioperm(0x220,16,1);
    outb(chn,port+10);  /* channel setting  */
    for (j=0; j<20000; j++);
    outb(0,port+12);  /* software triggering */
    printf("triggering \n");
        dh=inb(port + 5);
        dl=inb(port + 4);
        dta=dh*256 + dl-2048;

      ioperm(0x220,16,0);
      end_aq=time(a) - tz;
      volt=dta/409.6;

    printf("dta='%d' \n", dta);
    printf("V=%f \n", volt);
    printf("dh='%d' \n", dh);
    printf ("dl='%d' \n", dl);
/* Save data to disk    */

    if((fp=fopen(data, "at"))==NULL) {
        printf("cannot open file \n");
        return;
        }
        fprintf(fp,  "%d\t%.3f\t%d\n",st_aq,  volt,
        end_aq);
        fclose(fp);
        t0=t0+100;
  }
}
```

Program 1. Data aquisition program

ADC는 입력되는 아날로그 신호를 디지탈로 변환하여 특정 I/O 포트 레지스터를 변화시킨다. 컨버터의 종류와 정밀도에 따라 사용하는 주소와 레지스터의 수가 달라지므로 A/D 변환에 사용되는 주소와 레지스터에 관하여서는 컨버터 설명서를 참조하여야 한다. 여기서 사용한 PCL 711s는 8채널, 12bit ADC로 하나의 포트레지스터가 8bit 임으로 주소가 port+4와 port+5인 두 개의 레지스터를 사용한다. (port : I/O 포트 베이스 주소, 16 진수로 '220') port+4는 하

Table 1. Real time results of Program 1

| triggering time(sec) | Measured data(V) | after sampling |
|---|---|---|
| 100 | 2.952 | 100 |
| 200 | 2.952 | 200 |
| 300 | 2.954 | 300 |
| 400 | 2.954 | 400 |
| : | : | : |
| 59600 | 2.957 | 59600 |
| 59700 | 2.957 | 59700 |
| 59800 | 2.957 | 59800 |
| 59900 | 2.957 | 59900 |

위 바이트고 port+5는 상위 바이트인데 port+5의 경우는 아래 4개의 bit만 사용한다. 이 두 레지스터를 주기적으로 폴링(polling) 하여 주어진 변환식으로 계산하면 입력되는 아날로그 신호의 값을 알 수 있다. 한 채널에 대한 신호를 파일로 저장하는 프로그램(컴파일러 GNU C)을 프로그램 1에 나타내었다. 이 프로그램을 최적화 컴파일 하여 독립된 프로세스로 여러 개를 메모리에 올릴 수 있으므로 각 채널에 대해 독립적으로 데이터 수집을 할 수 있게 된다. Table. 1은 약 3V인 상용전지의 전압을 프로그램 1로 모니터링한 값이다. 이 프로세스는 CPU(central processing unit) 활동의 약 0.1%를 점유하며, CPU의 99%를 다른 작업에 사용하는 조건에서도 일정한 샘플링간격을 유지하였다. 이 프로그램에서는 트리거링(triggering)을 위하여 시간함수를 1초에 한 번씩 부르게 정하였고 시간(초)의 값이 정수단위이므로, 샘플링 간격과는 무관하게 실험시작시간 기준으로 최대 2초까지의 오차를 줄 수 있으나 각 샘플링 때마다 시간을 실험시작시간 기준으로 산출하므로 오차는 누적되지 않는다.

프로그램을 서버와 클라이언트 프로그램으로 나누어 놓는 것도 사용상 편리하다. 서버 프로세스는 ADC 각 채널의 값을 백그라운드로 샘플링하여 메모리의 어떤 공간에 쓰게 하고 클라이언트 프로세스는 필요한 채널의 값만 그 공유한

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <errno.h>
#include <asm/io.h>
#include <asm/unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <time.h>
#define ADC_PORT 0x220
#define LOG "/tmp/adc.log"
short shmid;
short *addr;
void shcall(len,cmds)
short len;
char *cmds;
{
 short smsize, smflag ;
 key_t keyno;
 int mypid ;
 smsize=len*sizeof (short);
 keyno=ftok(cmds,'a');
 printf("using key number %ld ...\n",keyno);
 if(shmid = shmget(keyno, len, 0777) >= 0) {
  printf("remove old shared memories\n");
  shmctl(shmid, IPC—RMID, NULL) ;
 }
 shmid=shmget(keyno, smsize, IPC_CREAT | 0777 |
 IPC—EXCL) ;
 if(shmid < 0)
 {
  printf("SHMGET error %d\n", errno) ;
  exit(1);
 }
 addr=(short *)shmat(shmid, (char*)0, 0) ;
 if((int)addr == -1 )
 {
  printf("SHMAT failed, error=%d\n", errno) ;
  shmctl(shmid, IPC_RMID, NULL) ;
  exit(1) ;
 }
 printf("shmem length %d with id %d my pid %d\
 n", len,shmid,mypid=getpid());
}
main(argc,argv)
int argc;
char *argv[];
{
 int port,i,j,dummy ;
 pid_t pid;
 port=ADC_PORT;
 printf("%s",argv[0]);
 printf(": ADC server ver. 0.0 \n");
 shcall(16,"/etc/hosts");
 switch(pid=fork())
/*  switch(pid=0) */
 {
  case 0:
  close(open("/dev/tty",O_RDWR));
  logger("start ADC daemon");
  ioperm(port,16,1);
  for(;;) {
  for(i=0;i < 8; i++){
  outb(i,port+10);/* channel setting 0 */
   for (j=0;j<20000; j++);
   outb(0,port+12);/* software triggering */
```

```
    while( addr[i*2] = inb(port +5) > 15 );  /*
    check D4 bit at DH */
    addr[i*2]=inb(port +5);
    addr[i*2+1]=inb(port + 4); /* DL */
    sleep(1);
   }
  }
  exit();
  case -1:printf("fail to fork\n"); exit(1);
  default:printf("Daemon started pid=%d\n",pid);
  exit();
  }
}
logger(msg)
char msg[];
{
time_t tloc;
FILE *log;
char ct[100];
/* Setup login file */
time(&tloc);
strcpy(ct,ctime(&tloc));
ct[strlen(ct)-1] = '\0';
log=fopen(LOG,"a");
fprintf(log,"%s %s\n",ct,msg);
fclose(log);
}
```

Program 2. Server program for ADC

```
#include <sys/ipc.h>
#include <sys/shm.h>
#include <errno.h>
short shmid;
short *addr;
key_t keyno;
main ()
{
 int i;
 keyno=ftok("/etc/hosts",'a');
 shmid=shmget(keyno, 32, 0777) ;
 addr=(short *)shmat(shmid, (char*)0, 0) ;
 if((int)addr == -1)
 {
  printf("SHMAT failed, error = %d\n", errno);
/* shmctl(shmid, IPC_RMID, NULL) ; */
   exit(1) ;
  }
   for(;;) {
      for(i=0; i <16; i++) printf("%4d", addr
      [i]);
      printf("\n");
      sleep(2);
  }
}
```

Program 3. Client program for ADC

메모리(shared memory)영역에서 가져다 사용한다. 클라이언트 프로세스는 ADC 디바이스에 직접 관계하지 않으므로 프로그램시 ADC를 사용함으로 해서 고려하여야 할 하드웨어적인 문제를 생각하지 않아도 되며 분산화로 프로그래밍이 단순해진다. 프로그램 2, 3에 간단한 서버와 클라이언트 프로그램을 나타내었다. 클라이언트 프로그램 수행시 나타나는 값은 측정된 값을 나타내는 두 레지스터(port+4, port+5)의 값이다.

여기에 소개한 프로그램 예를 이용하면 ADC 뿐 아니라 DAC (Digital to analog converter), Relay control device 등 각종 디바이스를 제어할 수 있어 시스템 자동화를 손쉽게 할 수 있다.

## 3. 결론 및 토의

PC 용 시 분할(time sharing) 운영체제를 사용하면 그 자체에서 제공하는 멀티테스킹(multi-tasking) 및 네트웍 기능이 지원되므로 이것을 이용하여 프로세스의 독립적 관리, 데이터 화일 핸들링 및 프로그래밍의 분산화가 용이하고, 측정시스템들을 통합적으로 관리할 수 있다.

이러한 기능에 대하여 IBM 호환 컴퓨터에 사용되는 다른 운영체제와 비교해 보면 도스의 경우 싱글테스킹이므로 이상의 모든 것이 가능하지 않고 윈도우와 OS/2의 경우 멀티테스킹(multi-tasking)이 가능하므로 프로세스를 일부 독립적으로 관리할 수 있으나 네트웍 상에서 서버시스템이 될 수 없으므로 그 효용은 반감하게 된다.

이 운영체제는 IBM 호환 PC를 기반으로 하기 때문에 하드웨어적으로 사용자 지향적인 확장성과, PC 용으로 개발된 모든 디바이스를 사용할 수 있으므로, 상용 웍스테이션에 비교가 안되는 적은 비용으로 시스템을 구축 및 유지관리할 수 있다. 또한 TCP/IP* 기반의 네트웍상에서 PC를 서버컴퓨터로 사용할 수 있으므로 Client/Server 응용프로그램(httpd**, Netscape*** 등)을 이용하면 부식측정상태를 네트웍(Network)을 통하여 실시간 관찰할 수 있게 하는 환경을 제공할 수도 있다.

여기에 소개한 운영체제 및 프로그램은 극히

짧은 시간에 많은 샘플링을 요하는 실험에는 적합하지 않다. 그리고 그러한 짧은 실험은 멀티테스킹 기능이 필요하지 않을 것이다. 그러나 장기적인 분극저항 모니터링, 전기화학적 wet/dry 실험, 장기 부식전위모니터링, 온도 및 습도 모니터링, 장기 정전위부식, 장기 갈바닉부식, 플랜트의 음극화 보호 등과 같이 오랜 시간이 요하는 실험에 유용할 것으로 생각한다.

*TCP/IP is a standard Transmission Control and Internet Protocol.

**Httpd is a network software partially developed at the University of Illinois Urbana-Champaign.

***Netscape is a trademark of Netscape Corporation.

## 5. 참 고 문 헌


1. 한국원자력연구소 연구보고서, KAERI-NEMAC/RR-36/91, p.12. (1992).
2. R.Brousseau, M.Arnott, and B.Baldock, Corrosion, **51**, No. 8, 639 (1995).
3. H.Malik, Corrosion, **51**, No.4, 321 (1995).
4. 한국원자력연구소 연구보고서, KAERI-NEMAC/RR-131/94, p.15. (1994).
5. 박수문, 변종홍, "실험실에서의 소형전산기", 자유아카데미,1989.
6. B.Yang, Corrosion, **51**, No.2, 153 (1995).
7. L. Kobotiatis, C.Tsikrika, and P.G.Koutsoukos, Corrosion, **51**, No.1, 19 (1995).
8. Matt Welsh, "Linux Installation and Getting Started", Specialized System Consultants, 1994.
9. Michael K. Johnson, "Linux kernel Hackers' Guide", in Slackware Reference Guide, 1993.
10. R.Baruch and C.Schroeter, "Writing Character Device Driver for Linux", in Slackware Reference Guide, 1994.