과제 #1

홍경인

M1522.006700 확장형 고성능 컴퓨팅 (001)

September 27, 2024

1 Compilation Process

1.1 Preprocessing

(a) `stdio.h`는 로그인 노드의 `root/usr/include`에 있으며, 파일의 라인 수는 876이다. `math.h`는 로그인 노드의 `root/usr/include`에 있으며, 파일의 라인 수는 1342이다.

(b) `gcc -E sqrt.c > sqrt_preprocess.c`로 preprocess를 완료한 뒤, 아래와 같이 `scanf`, `printf`, `sqrt`가 나타난 모든 줄을 출력받았다.

```
>>> shpc159@elogin3:~$ grep 'scanf' sqrt_preprocess.c
extern int fscanf (FILE *__restrict __stream,
extern int scanf (const char *__restrict __format, ...) ;
extern int sscanf (const char *__restrict __s,
extern int fscanf (FILE *__restrict __stream, const char *__restrict __format,
    ...) __asm__ ("" "__isoc99_fscanf")
extern int scanf (const char *__restrict __format, ...) __asm__ ("" 
    "__isoc99_scanf")
extern int sscanf (const char *__restrict __s, const char *__restrict __format,
    ...) __asm__ ("" "__isoc99_sscanf") __attribute__ ((__nothrow__ , __leaf__))
extern int vfscanf (FILE *__restrict __s, const char *__restrict __format,
     __attribute__ ((__format__ (__scanf__, 2, 0))) ;
extern int vscanf (const char *__restrict __format, __gnuc_va_list __arg)
     __attribute__ ((__format__ (__scanf__, 1, 0))) ;
extern int vsscanf (const char *__restrict __s,
     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__format__
         (__scanf__, 2, 0)));
extern int vfscanf (FILE *__restrict __s, const char *__restrict __format,
    __gnuc_va_list __arg) __asm__ ("" "__isoc99_vfscanf")
     __attribute__ ((__format__ (__scanf__, 2, 0))) ;
```

```
extern int vscanf (const char *__restrict __format, __gnuc_va_list __arg) __asm__
    ("" "__isoc99_vscanf")
     __attribute__ ((__format__ (__scanf__, 1, 0))) ;
extern int vsscanf (const char *__restrict __s, const char *__restrict __format,
    __gnuc_va_list __arg) __asm__ ("" "__isoc99_vsscanf") __attribute__
    ((__nothrow__ , __leaf__))
     __attribute__ ((__format__ (__scanf__, 2, 0)));

>>> shpc159@elogin3:~$ grep 'printf' sqrt_preprocess.c
extern int fprintf (FILE *__restrict __stream,
extern int printf (const char *__restrict __format, ...);
extern int sprintf (char *__restrict __s,
extern int vfprintf (FILE *__restrict __s, const char *__restrict __format,
extern int vprintf (const char *__restrict __format, __gnuc_va_list __arg);
extern int vsprintf (char *__restrict __s, const char *__restrict __format,
extern int snprintf (char *__restrict __s, size_t __maxlen,
     __attribute__ ((__nothrow__)) __attribute__ ((__format__ (__printf__, 3, 4)));
extern int vsnprintf (char *__restrict __s, size_t __maxlen,
     __attribute__ ((__nothrow__)) __attribute__ ((__format__ (__printf__, 3, 0)));
extern int vdprintf (int __fd, const char *__restrict __fmt,
     __attribute__ ((__format__ (__printf__, 2, 0)));
extern int dprintf (int __fd, const char *__restrict __fmt, ...)
     __attribute__ ((__format__ (__printf__, 2, 3)));
  printf("Usage: ./sqrt number\n");
  printf("Example: ./sqrt 2\n");
void print_sqrt(double number) { printf("%.8lf\n", sqrt(number)); }

>>> shpc159@elogin3:~$ grep 'sqrt' sqrt_preprocess.c
# 1 "sqrt.c"
# 1 "sqrt.c"
extern double sqrt (double __x) __attribute__ ((__nothrow__ , __leaf__)); extern
    double __sqrt (double __x) __attribute__ ((__nothrow__ , __leaf__));
extern float sqrtf (float __x) __attribute__ ((__nothrow__ , __leaf__)); extern
    float __sqrtf (float __x) __attribute__ ((__nothrow__ , __leaf__));
extern long double sqrtl (long double __x) __attribute__ ((__nothrow__ ,
    __leaf__)); extern long double __sqrtl (long double __x) __attribute__
    ((__nothrow__ , __leaf__));
# 2 "sqrt.c" 2
# 3 "sqrt.c" 2
# 4 "sqrt.c" 2
# 5 "sqrt.c"
  printf("Usage: ./sqrt number\n");
  printf("Example: ./sqrt 2\n");
```

```
void print_sqrt(double number) { printf("%.8lf\n", sqrt(number)); }
  print_sqrt(atof(argv[1]));
```

(c) 위의 결과에는 실제로 scanf, printf, sqrt을 구현하는 코드가 존재하지 않는다. 해당 함수는 extern 형태로 호출되고 있으며 이는 함수가 외부 파일에 저장되어 있음을 나타낸다. 그 이유는 preprocess 시 헤더 파일 내 함수의 코드가 직접 추가되는 것이 아니라, 함수의 프로토타입만이 추가되기 때문이다.

1.2 Compilation

(a) gcc -c sqrt.c로 sqrt.o를 생성할 수 있었다.

(b) file sqrt.o 명령을 실행하여 아래와 같은 출력을 얻는다.

```
sqrt.o: ELF 64-bit LSB relocatable, x86-64, version 1 (SYSV), not stripped
```

이로부터 파일 포맷이 ELF(Executable and Linkable Format)임을 안다. 이는 Linux에서 생성하는 object file의 포맷이다.

1.3 Linking

(a) gcc sqrt.o와 같이 시도하면 C 표준 라이브러리에 한해 linking이 이루어진다. 하지만 sqrt는 헤더 파일에서 extern으로 선언되어 있으므로 linking 시 C 표준 라이브러리가 아닌 다른 라이브러리와 연결해 주어야 한다. 구체적으로 sqrt는 math.h가 포함하는 다른 헤더 파일(mathcalls.h)에서 아래와 같이 선언된다.

```
__MATHCALL (sqrt,, (_Mdouble_ __x));
```

이때 사용되는 매크로 __MATHCALL은 직접적으로 정의되지 않으나, 매크로 간의 관계를 통해 해당 함수를 extern으로 선언하고 있음을 알 수 있다.

외부 라이브러리를 연결하기 위해서 -lm 옵션을 추가할 필요가 있는데, -l은 외부 라이브러리를 linking하도록 하는 옵션이고, 이때 뒤에 오는 알파벳이 lib을 제외한 라이브러리의 이름이 된다. 함수 sqrt는 서버 컴퓨터 기준 /lib/x86_64-linux-gnu/libm.so.6에 구현되어 있는데, 이때 라이브러리의 이름이 libm이므로 m을 옵션에 추가한다. 따라서 최종 실행 파일은 다음 명령어를 통해 실행되어야 한다.

```
gcc sqrt.o -o sqrt -lm
```

(b) 위의 명령어와 같이 컴파일을 마무리하면 sqrt 파일을 얻을 수 있다. 이를 알맞게 실행하여 아래와 같은 결과를 얻었다.

2 C Programming

2.1 Shift

(a) -16을 32-bit 2의 보수표현으로 나타낸 binary number는 다음과 같다.

0b11111111111111111111111111110000

(b) `a >> 2`에서는 arithmetic shift가 이루어지므로 그 결과는 다음과 같다.

0b11111111111111111111111111111100

(c) `ua >> 2`에서는 logical shift가 이루어지므로 그 결과는 다음과 같다.

0b00111111111111111111111111111100

(d) Right shift 하는 상황에서, Arithmetic Shift는 빈 bit에 MSB를 저장하여 Shift하는 것이고 Logical Shift는 빈 bit에 단순히 0을 저장하여 Shift하는 것이다. 따라서 Arithmetic Shift가 일어난 (b)에서는 MSB인 1이 좌측에 추가되었고, Logical Shift가 일어난 (c)에서는 0이 추가되었다.

2.2 Convert

(a) `convert.c` 와 같이 코드를 작성하였다.

3 클러스터 사용 연습

다음은 문제의 명령을 9월 26일 (목) 오전 12시 15분경에 실행하여 얻은 결과와 그 의미이다.

(a) `sinfo` 명령을 수행한 결과는 아래와 같다.

```
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
class1       up       2:00      1    mix a04
class1       up       2:00      3  alloc a[00-02]
class1       up       2:00      8   idle a[03,05-11]
```

`PARTITION`은 특정 사용자 집단이 쓸 수 있도록 미리 구획된 노드 집합이다. 주어진 결과에서 모든 노드가 동일한 `PARTITION`으로 구획되어 있음을 확인할 수 있다. `AVAIL`은 노드의 사용 가능성과 작업 할당 가능성을 나타낸다. 현재는 모든 노드가 `up` 상태에 있어

노드를 사용할 수 있고 작업이 할당될 수 있으나, 노드를 사용할 수 없고 작업이 할당될 수도 없는 down과 기존에 할당된 작업이 노드를 사용하나 신규 작업 할당이 불가한 drain, 어떤 작업도 노드를 사용할 수 없고 신규 작업 할당도 불가한 inactive 상태 역시 존재한다.

TIMELIMIT은 각 노드에서 실행이 가능한 최대 시간을 나타내며, 현재는 hour:minute 포맷을 사용하고 있으므로 2시간이다. NODES는 각 행에 해당되는 노드 수를 나타내고 STATE는 노드의 상태를 다양하게 나타내며, mix는 노드의 리소스 중 일부가 작업에 할당되어 있음을 나타내고 alloc은 노드가 현재 작업에 할당되어 사용할 수 없음을, idle은 노드가 할당된 작업이 없어 사용할 수 있음을 나타낸다.

(b) squeue 명령을 수행한 결과는 아래와 같다.

```
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

이는 작업의 ID JOBID, 작업이 제출된 파티션 PARTITION과 작업 이름 NAME과 사용자 이름 USER에 대해 작업이 특정 상태 ST에 있으며 TIME 시간 동안 실행되고 있고 NODES 만큼 노드가 할당되어 있음을 나타낸다. 이때 NODELIST(REASON)은 특정 작업에 할당된 노드의 종류나, 특정 작업이 시작되지 못한 경우 그 이유가 무엇인지 알려 준다.

현재는 아무 작업도 대기 중이거나 실행 중이지 않으므로 결과가 비어 있다.

(c) srun -N 2 hostname 명령을 수행한 결과는 아래와 같다.

```
srun: job 824269 queued and waiting for resources
srun: job 824269 has been allocated resources
a06
a05
```

명령어는 slurm에서 hostname 명령을 수행할 것을 지시하고 있다. hostname은 각 노드의 이름을 출력하는 명령으로, 노드 두 개를 쓰도록 지정하고 있으므로 두 노드의 이름이 출력되어야 한다. 수행한 결과에서 a06, a05가 차례대로 출력되고 있으므로 명령이 잘 수행되었음을 알 수 있다.

(d) 다음은 lscpu 명령을 실행한 결과이다. 이는 CPU 아키텍처와 관련된 정보를 반환하도록 지시하는 명령어이다. 따라서 아래와 같이 로그인 노드의 CPU 정보가 출력된다.

```
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
Address sizes: 46 bits physical, 48 bits virtual
CPU(s): 32
On-line CPU(s) list: 0-31
```

```
Thread(s) per core: 2
Core(s) per socket: 8
Socket(s): 2
NUMA node(s): 2
Vendor ID: GenuineIntel
CPU family: 6
Model: 79
Model name: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
Stepping: 1
CPU MHz: 1200.613
CPU max MHz: 3000.0000
CPU min MHz: 1200.0000
BogoMIPS: 4199.96
Virtualization: VT-x
L1d cache: 512 KiB
L1i cache: 512 KiB
L2 cache: 4 MiB
L3 cache: 40 MiB
NUMA node0 CPU(s): 0-7,16-23
NUMA node1 CPU(s): 8-15,24-31
Vulnerability Itlb multihit: KVM: Mitigation: Split huge pages
Vulnerability L1tf: Mitigation; PTE Inversion; VMX conditional cache flushes, SMT
    vulnerable
Vulnerability Mds: Mitigation; Clear CPU buffers; SMT vulnerable
Vulnerability Meltdown: Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via
    prctl and seccomp
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer
    sanitization
Vulnerability Spectre v2: Mitigation; Full generic retpoline, IBPB conditional,
    IBRS_FW, STIBP conditional, RSB filling
Vulnerability Srbds: Not affected
Vulnerability Tsx async abort: Mitigation; Clear CPU buffers; SMT vulnerable
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
    clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
                                rdtscp lm constant_tsc arch_perfmon pebs bts rep_good
                                    nopl xtopology nonstop_tsc cpuid aperfmperf pni
                                    pclmulqdq dtes64 monitor ds_cpl vmx s
                                mx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca
                                    sse4_1 sse4_2 x2apic movbe popcnt
                                    tsc_deadline_timer aes xsave avx f16c rdrand
                                    lahf_lm ab
                                m 3dnowprefetch cpuid_fault epb cat_l3 cdp_l3
```

```
                                    invpcid_single pti intel_ppin ssbd ibrs ibpb
                                    stibp tpr_shadow vnmi flexpriority ept vpid ept_
                                ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms
                                    invpcid rtm cqm rdt_a rdseed adx smap intel_pt
                                    xsaveopt cqm_llc cqm_occup_llc cqm_mbm_t
                                otal cqm_mbm_local dtherm ida arat pln pts md_clear
                                    flush_l1d
```

`srun -N 1 lscpu`는 `lscpu`을 slurm의 `srun` 명령어를 통해 계산 노드로 전송하는 명령어이다. 따라서 계산 노드의 CPU 정보가 출력된다. 로그인 노드와 계산 노드의 CPU 사양이 다르므로, 출력 결과가 위의 `lscpu`의 결과와 다르다.

```
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
Address sizes: 46 bits physical, 48 bits virtual
CPU(s): 32
On-line CPU(s) list: 0-31
Thread(s) per core: 2
Core(s) per socket: 8
Socket(s): 2
NUMA node(s): 2
Vendor ID: GenuineIntel
CPU family: 6
Model: 79
Model name: Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz
Stepping: 1
CPU MHz: 1200.613
CPU max MHz: 3000.0000
CPU min MHz: 1200.0000
BogoMIPS: 4199.96
Virtualization: VT-x
L1d cache: 512 KiB
L1i cache: 512 KiB
L2 cache: 4 MiB
L3 cache: 40 MiB
NUMA node0 CPU(s): 0-7,16-23
NUMA node1 CPU(s): 8-15,24-31
Vulnerability Itlb multihit: KVM: Mitigation: Split huge pages
Vulnerability L1tf: Mitigation; PTE Inversion; VMX conditional cache flushes, SMT
    vulnerable
Vulnerability Mds: Mitigation; Clear CPU buffers; SMT vulnerable
Vulnerability Meltdown: Mitigation; PTI
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass disabled via
```

```
    prctl and seccomp
Vulnerability Spectre v1: Mitigation; usercopy/swapgs barriers and __user pointer
    sanitization
Vulnerability Spectre v2: Mitigation; Full generic retpoline, IBPB conditional,
    IBRS_FW, STIBP conditional, RSB filling
Vulnerability Srbds: Not affected
Vulnerability Tsx async abort: Mitigation; Clear CPU buffers; SMT vulnerable
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
    clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
                                rdtscp lm constant_tsc arch_perfmon pebs bts rep_good
                                    nopl xtopology nonstop_tsc cpuid aperfmperf pni
                                    pclmulqdq dtes64 monitor ds_cpl vmx s
                                mx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid dca
                                    sse4_1 sse4_2 x2apic movbe popcnt
                                    tsc_deadline_timer aes xsave avx f16c rdrand
                                    lahf_lm ab
                                m 3dnowprefetch cpuid_fault epb cat_l3 cdp_l3
                                    invpcid_single pti intel_ppin ssbd ibrs ibpb
                                    stibp tpr_shadow vnmi flexpriority ept vpid ept_
                                ad fsgsbase tsc_adjust bmi1 hle avx2 smep bmi2 erms
                                    invpcid rtm cqm rdt_a rdseed adx smap intel_pt
                                    xsaveopt cqm_llc cqm_occup_llc cqm_mbm_t
                                otal cqm_mbm_local dtherm ida arat pln pts md_clear
                                    flush_l1d
```