Министерство цифрового развития, связи и массовых коммуникаций
Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
Кафедра Вычислительных систем (ВС)

ОТЧЕТ
о практической работе
«Исследование иерархии памяти, работы процессора в ЭВМ на языке С»

Работу выполнил:
студент 1 курса
группы ИС-541
Устюжанин Д.К.

г. Новосибирск
2025 г.

Содержание

Введение

Цель работы: Научиться получать уникальные характеристики процессора и ОС Linux через /proc/cpuinfo, /proc/meminfo. Понять, как иерархия памяти влияет на производительность. Написать программу, которая корректно работает только на домашней системе, используя её уникальные параметры (модель CPU, объём RAM и т.п.). Закрепить навыки работы с буферизованным вводом (getchar, putchar) и циклическими вычислениями.

Теоретическая часть

Полная информация о процессоре в операционных системах Linux хранится в файле `/proc/cpuinfo`. Также дополнительную информацию о процессоре можно узнать с помощью команды `lspu`. Последний способ позволяет получить размер кэш-памяти всех трех уровней (L1 (во многих современных процессорах L1d для хранения данных и L1i для хранения информации разделены), L2, L3) в удобном виде. L1-кэш является самым быстрым из всех, но самым меньшим по объему памяти, хранится в каждом ядре процессора. L2-кэш менее быстрый, имеет объем памяти больше, чем L1-кэш, расположен на кристалле процессора и имеется у каждого ядра. L3-кэш наименее быстрый, но с самым большим объемом памяти, хранится на кристалле процессора, но дальше от ядер, чем L2-кэш, является общим для всех ядер.

Пример получения информации о имени модели процессора, количестве ядер и кэш-памяти процессора с помощью чтения файла `/proc/cpuinfo`:

```
cat /proc/cpuinfo | grep "model name\|cpu cores\|cache size" | head -n 3
```

Пример вывода:

```
model name : 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz
cache size : 12288 KB
cpu cores : 6
```

Пояснения:

Команда `cat` читает файл и выводит содержимое на экран, далее с помощью pipe (символа «`|`») содержимое передается на вход команде `grep`, которая ищет строки, содержащие выражения «`model name`», или (символ «`|`» означает логическое «или») «`cpu cores`», или «`cache size`», после чего содержимое передается на вход команде `head`, которая читает первые 3 строки содержимого (`-n 3`). Последнее нужно для того, чтобы не выводить одинаковые строки, так как в `/proc/procinfo` содержится информация по каждому ядру процессора, то есть, если процессор, например,

шестиядерный, то в файле указано по 3 одинаковые строки (необходимые для вывода) для каждого ядра (имя модели, размер кэш-памяти и количество ядер).

В результате выполнения этих команд были получены три строки: имя модели процессора: `11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz`; размер кэш-памяти процессора, соответствующий L3-кэшу (доказательство ниже) в килобайтах: `12288 KiB`; количество ядер процессора: `6`.

Пример получения информации о размерах кэш-памяти всех трех уровней:

```
lscpu | grep cache
```

Пример вывода:

```
L1d cache:  288 KiB (6 instances)
L1i cache:  192 KiB (6 instances)
L2 cache:  7.5 MiB (6 instances)
L3 cache:  12 MiB (1 instance)
```

Пояснения:

Команда `lscpu`, как было сказано выше, выводит некоторую информацию о процессоре. Она была переправлена с помощью pipe на вход команде `grep`, которая ищет строки, содержащие слово «`cache`».

В результате выполнения команд были получены размеры всех кэшей: L1d-кэш, хранящий данные, равен 288 КБ; L1i-кэш, хранящий инструкции, равен 192 КБ; L2-кэш равен 7,5 МБ; L3-кэш равен 12 МБ. В скобках рядом с размером каждого кэша написано количество физических блоков кэш-памяти. Например, для L3-кэша: `L3 cache: 12 MiB (1 instance)`. Это означает, что ячейка в 12 МБ является общей для всех ядер.

Для получения информации об оперативной памяти в операционных системах Linux существует файл `/proc/meminfo`, схожий с `/proc/cpuinfo`.

Пример получения информации об оперативной памяти:

```
cat /proc/meminfo | grep MemTotal
```

Пример вывода:

```
MemTotal:  16103368 kB
```

Пояснения:

Таким же способом, как и раннее, происходит чтение файла с нужными данными с помощью `cat`, перенаправление с помощью pipe в `grep` и поиск фразы `MemTotal`. Таким образом был получен размер всей имеющейся оперативной памяти равный `16103368 kB`.

Иерархия памяти в современной ЭВМ представляет собой многоуровневую структуру, где, чем ближе к процессору, тем быстрее, но меньше объем. Данная иерархия включает следующие уровни, расположенные в порядке убывания скорости доступа и возрастания объёма: **регистры** процессора находятся на вершине иерархии как самые быстрые, но и самые малые по объёму элементы памяти, непосредственно встроенные в процессор; кэш-память **L1** разделяется на кэш инструкций и кэш данных, обеспечивая максимальную скорость доступа к наиболее часто используемой информации; кэш-память **L2** обычно представляет собой общий кэш для инструкций и данных, служащий буфером между L1 и более медленными уровнями памяти; кэш-память **L3** функционирует как общий кэш для всех ядер процессора, координируя обмен данными между различными вычислительными ядрами; оперативная память (**RAM**) составляет основную рабочую память системы, обеспечивая хранение исполняемых программ и обрабатываемых данных; виртуальная память, реализуемая через swap-раздел на жёстком диске, представляет самый медленный, но практически неограниченный уровень иерархии памяти.

Кэш-память эффективно компенсирует значительную разницу в скорости работы между центральным процессором и оперативной памятью благодаря реализации нескольких фундаментальных принципов организации вычислительных систем. Принцип локальности лежит в основе работы кэш-памяти и проявляется в двух основных формах. Временная локальность означает, что если к определённым данным или инструкциям было произведено обращение, высока вероятность, что они понадобятся снова в ближайшем будущем. Пространственная локальность предполагает, что при обращении к определённому адресу памяти, с большой вероятностью вскоре понадобятся данные из соседних адресов. Иерархическая

структура кэш-памяти организована таким образом, что каждый последующий уровень служит буфером для следующего, более медленного уровня. Кэш L1, находящийся непосредственно в ядре процессора, обеспечивает наивысшую скорость доступа при минимальном объёме. Кэши L2 и L3, будучи более медленными, предлагают значительно больший объём для хранения данных. Эффективность работы кэш-памяти определяется высоким процентом попаданий в кэш (cache hit), который в современных процессорах достигает 90-95%. При попадании данные извлекаются из быстрой кэш-памяти, а при промахе (cache miss) происходит обращение к более медленной оперативной памяти с параллельным сохранением полученных данных в кэше для возможных будущих обращений.

Программная часть

Листинг программы:

```
#include <stdio.h>
#include <ctype.h> // isdigit
#include <math.h> // pow

#define MEM 65536
#define DATA 64*1024 // L1d = 288KiB (6 intenses) => 48KiB/core

char memory[MEM];
char data[DATA];

// reads first number from stdout
void geti(int*num) {
  char s[21]; // max number of digits + 1 for int 64-bit
  char *p = s;
  char ch;
  int count=0; // counts numbers in input
  // reads only numbers
  while ((ch=getchar()) != ' ' &&
    ch!='\n' && ch!='\t' && ch!=EOF && isdigit(ch)) {*p++ = ch; count
++;}

  p = s; // point at start of input string
  // convert chars to number
  for (int i=pow(10, count-1); i>0; i/=10, p++) *num += (*p-'0')*i;
}

// prints number
void puti(int num, char *ch) {
  char s[21];
  char *p = s;
  // reads digits in reverse
```

```
  for (int n=num; n > 0; n/=10) *p++ = ('0' + n%10);
  int len = p - s;
  p = ch;
  // prints in normal way with encryption
  for (int i=len-1; i>=0; i--) {
    putchar((*(s+i)-'0' + *(p++))%10 + '0');
  }
  putchar('\n');
}

// reads a line
void getl(char *s) {
  char ch;
  char *p=s;
  while ((ch=getchar()) != ' ' && ch!='\n' && ch!='\t' && ch!=EOF)
*p++ = ch;
  *p='\0';
}

int main() {
  setbuf(stdin, NULL); // no-buff stdin

  int N=0, sum=0;
  char proc_model[20] = {0};

  getl(proc_model);
  geti(&N);
  for (int i=1; i<=N; i++) {
    sum += i;
    memory[i%MEM] = i%256;
    data[i%DATA] = (sum+i)%256;
  }
  puti(sum, proc_model);
```

```
}
```

Описание работы программы:

Нагрузка на процессор создаётся за счёт выполнения цикла от 1 до N с арифметическими операциями сложения, операций взятия модуля для работы с массивами и преобразования данных между различными форматами. Каждая итерация цикла включает несколько вычислительных операций, что создаёт значительную нагрузку на центральный процессор, особенно при больших значениях N. Нагрузка на память обеспечивается использованием двух массивов: `memory[65536]` и `data[65536]`, а также постоянной записью данных в массивы по циклическому принципу. Обработка данных, превышающих размер кэш-памяти L1, приводит к интенсивному использованию оперативной памяти и системы кэширования.

Привязка к уникальным характеристикам системы реализована через чтение модели процессора из входного потока и использование названия процессора как ключа для шифрования результата. Программа считывает модель процессора через функцию `getl()` и сохраняет её в массив `proc_model`. При изменении модели процессора результат шифрования будет отличаться, что обеспечивает привязку программы к конкретной системе. Данный механизм делает программу уникальной для каждой конфигурации оборудования, поскольку шифрование результата напрямую зависит от идентификатора процессора.

Работа с кэш-памятью организована таким образом, что размер массива `data[65536]` (64 КБ) превышает типичный размер кэша L1 (32-48 КБ), что специально спроектировано для создания промахов кэша. Циклическая запись в массив через операцию модуля обеспечивает случайный доступ к памяти, который плохо предсказуем для систем кэширования. Это приводит к частым промахам кэша (cache misses) и вынуждает процессор обращаться к более медленным уровням памяти, демонстрируя на практике влияние иерархии памяти на производительность.

Процесс шифрования результата начинается с преобразования вычисленной суммы в строку цифр. Каждая цифра затем последовательно складывается с ASCII-

кодами символов модели процессора, причём результат берётся по модулю 10 для получения конечной цифры.Зашифрованные цифры выводятся через функцию `putchar()`, что соответствует требованиям задания по использованию буферизованного вывода. Конкретный пример шифрования для модели процессора "i7-9700K" и суммы 123 выглядит следующим образом: первая цифра 1 складывается с ASCII-кодом 'i' (105), результат (106) берётся по модулю 10, получается 6; вторая цифра 2 складывается с кодом '7' (55), результат (57) по модулю 10 даёт 7; третья цифра 3 складывается с кодом '-' (45), результат (48) по модулю 10 даёт 8. Итоговый зашифрованный результат - "678".

Пример выполнения программы (из приложения №1):

Пример запуска:

```
echo "i5-11400H\n16" | main
```

Пример вывода:

```
661
```

Комментарии:

Запуск программы в примере происходит через перенаправление вывода команды `echo` на ввод программе (исполняемому файлу `main`). Естественно, программу можно запускать, как с перенаправлением, разделяя имя процессора и число N символом новой строки «`\n`», так и обычным запуском программы и вводом имени процессора и числа N с клавиатуры, разделенных нажатием клавиши `<Enter>` для ввода символа новой строки и этой же клавишей в конце либо вводом символа конца файла (`Ctrl-D` для Linux и `Ctrl-Z` для Windows).

Вывод

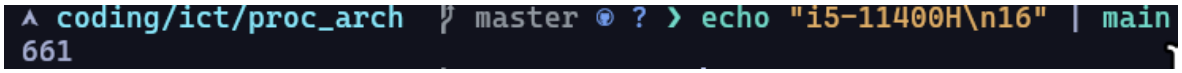
Программа работает корректно только на исходной машине благодаря механизму привязки к уникальным характеристикам системы. Ключевым элементом защиты является использование модели процессора в качестве криптографического ключа для шифрования результатов вычислений. При переносе исполняемого файла на другую машину с отличной моделью процессора алгоритм шифрования будет использовать иной набор ASCII-кодов, что приведёт к генерации совершенно другого зашифрованного результата. Даже при идентичных входных данных и одинаковом алгоритме вычислений, различие в криптографических ключах сделает результаты несовместимыми между системами.

Иерархия памяти оказывает фундаментальное влияние на производительность через механизм кэширования данных. Практическое исследование с массивом `data[65536]` продемонстрировало, что при работе с объёмами данных, превышающими размер кэша L1, происходит значительное увеличение количества промахов кэша. Каждый промах кэша приводит к необходимости обращения к более медленным уровням памяти (L2, L3, RAM), что создаёт задержки в выполнении программы. Оптимизация работы с памятью, включая учёт размеров кэш-памяти и принципов локальности данных, позволяет существенно повысить эффективность вычислительных процессов и сократить время выполнения программ.

Буферизация ввода представляет собой важный механизм оптимизации ввода-вывода, который значительно влияет на взаимодействие программы с пользователем. Использование функции `setbuf(stdin, NULL)` для отключения буферизации позволило наблюдать разницу в поведении программы. При отключённой буферизации программа обрабатывает ввод посимвольно, что обеспечивает немедленную реакцию на каждое нажатие клавиши. В режиме с буферизацией ввод накапливается в буфере до нажатия `<Enter>`, что уменьшает количество системных вызовов и повышает общую эффективность ввода-вывода, но задерживает обработку данных. Практическое применение показало, что выбор стратегии буферизации зависит от конкретных

требований приложения - либо немедленная обработка ввода, либо оптимизация производительности через уменьшение системных вызовов.

Приложение



Приложение №1 – Пример выполнения программы (ввод, запуск, вывод).