|  |
| --- |
|  |
| 1. 实验目的及实验内容   （本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析） |
| **实验目的：**  设计随机数发生器（真/假）  **实验内容：**  设计真随机数发生器，并至少通过NIST检测。 |
| 1. 实验环境及实验步骤   （本次实验所使用的器件、仪器设备等的情况；具体的实验步骤） |
| **实验环境：**  Visual studio 2019  VMwareWorkstationPro 15.5.0  Ubuntu 12.04.5 desktop i386 32位  **关键技术：**  熵生成：  通过汇编指令rdseed获取intel Broadwell架构的 CPU上硅片的热噪声作为随机数发生器的种子  熵提取（对获取的噪声进行变化）  通过Xorshift，梅森旋转，SHA-1算法对种子进行变化，输出生成的随机数。为真随机数发生器。  **实验步骤：**   1. **概述随机数**   常用的伪随机数的生成用的是模余法  Rn=(Rn−1∗A+C)modBRn=(Rn−1∗A+C)modB  A，B，CA，B，C都是常数，RnRn为产生的随机数，Rn−1Rn−1为上一拍的随机数，可以看出，这种算法产生的随机数并不足够随机，与初始状态有关，初始状态定了，随机序列也就定了。  真随机数顾名思义就是真正的随机数，比如我们掷骰子，产生的就是真正的随机数。有很多很天然的随机数，比如热噪声、键盘的输入、鼠标的位置，这些都是随机的，但是这些随机的信号直接拿来做随机数也不一定很随机，比如热噪声在频谱上并不是白噪声，而可能是有色噪声，鼠标的位置前后会有关联，一段时期可能在一个区域，所以这些随机信号并不能直接拿来作为随机数，而是需要经过处理。   1. **熵生成（生成seed）**   熵提取即为获取噪声，转化为可以处理的数据形式，作为真随机数生成器的seed。  通过汇编指令rdseed获取intel Broadwell架构的 CPU上硅片的热噪声作为随机数发生器的种子  GCC 4.6+和Clang 3.2+提供了RdRand的内置支持  使用visual studio，C++中嵌入\_asm，将seed存储到eax寄存器，然后读取到32位无符号整型中。  实现：  先判断CPU是否支持RDSEED  \_\_asm {  mov eax, 1  cpuid  and eax, 0x20000000 //30th bit  test eax, 0  jnz L  mov flag, 1  L:  }  if (!flag) {  cout << "Not support this CPU!";  return 0;  }  结果支持，生成seed。Seed为32位无符号整型，存到unsigned long result。  unsigned long result;  \_\_asm {  rdseed eax  mov result, eax  }   1. **熵提取（处理seed，生成随机数）**    1. **Xorshift得32位无符号整数**   使用上述方法获得三个32位长的seed，然后将这三个32位整数进行Xorshift变换，生成新的一个32位整数，增加随机性。  Xorshift原理为将三个32位长的字处理，拼接，形周期为2^96-1的线性变换。采用分块矩阵的方式，将32阶矩阵扩展为96维矩阵，(x, y, z)T = (y, z, xA + yC + zB)。其中 x, y, z 是三个 32 维向量，而 A、B、C 则是一两个这样的异或与乘法的迭代的复合。在计算机中通过移位和异或实现线性变换。    实现：  unsigned long result中存储着下一步梅森旋转所需的32位数。  unsigned long xorshf96(unsigned long x, unsigned long y, unsigned long z) { unsigned long t;  x ^= x << 16;  x ^= x >> 5;  x ^= x << 1;  t = x;  x = y;  y = z;  z = t ^ x ^ y;  return z;  }  result = xorshf96(result1, result2, result3);   * 1. **梅森旋转得32位无符号整数**   将3.1中获取的32位result，进行梅森旋转，得到新的32位整数，增加随机性。  梅森旋转算法（Mersenne twister），可以快速产生高质量的伪随机数，修正了古典随机数发生算法的很多缺陷。常见的两种为基于32位的 MT19937和基于64位的 MT19937-64。  梅森旋转算法是利用**线性反馈移位寄存器（LFSR）**产生随机数的，而MT19937梅森旋转算法的周期为 2^19937−1（正如算法名，这是一个梅森素数），说明它是一个19937级的线性反馈移位寄存器，梅森旋转算法是利用线性反馈寄存器一直进行移位旋转。  实现 :  class MT {  public:  int index;  unsigned long MT[624]; //624 \* 32 - 31 = 19937  void initial(unsigned long seed)  {  //printf("seed:%d\n", seed);  index = 0;  MT[0] = seed;  //对数组的其它元素进行初始化  for (int i = 1; i < 624; i++)  {  unsigned long t = 1812433253 \* (MT[i - 1] ^ (MT[i - 1] >> 30)) + i;  MT[i] = t & 0xffffffff; //取最后的32位赋给MT[i]  }  }  void twist() //遍历旋转链，进行旋转算法  {  for (int i = 0; i < 624; i++)  {  // 2^31 = 0x80000000  // 2^31-1 = 0x7fffffff  unsigned long y = (MT[i] & 0x80000000) + (MT[(i + 1) % 624] & 0x7fffffff);  MT[i] = MT[(i + 397) % 624] ^ (y >> 1);  if (y & 1)  MT[i] ^= 2567483615;  }  }  unsigned long Rand()  {  if (index == 0)  twist();  unsigned long y = MT[index];  y = y ^ (y >> 11); //y右移11个bit  y = y ^ ((y << 7) & 2636928640); //y左移7个bit与2636928640相与，再与y进行异或  y = y ^ ((y << 15) & 4022730752); //y左移15个bit与4022730752相与，再与y进行异或  y = y ^ (y >> 18); //y右移18个bit再与y进行异或  index = (index + 1) % 624;  return y;  }  void begin(unsigned long seed)  {  //rfile.open("C:\\Users\\lenovo\\Desktop\\rands.txt", ios::out | ios::app);  //if (!rfile.is\_open()) return ;  initial(seed); //初始化梅森旋转链  int cnt = 0;  unsigned long ret = 0;  string a;  ret = Rand();  ret = Rand(); //32位无符号整型  SHA1 sha;  sha.begin(ret); //调用3.3中的SHA-1  }  };   * 1. **SHA-1算法得5位随机数**   梅森旋转得到32位无符号整型后，作为message，进行sha-1签名，增加随机性。得到160位散列，然后取第32,64,96,128,160位，得到5位随机数，作为最终结果中的5位。  SHA-1对message分组，512位为一组，生成160位散列值。**SHA-1**（Secure Hash Algorithm 1，中文名：安全散列算法1）是一种[密码散列函数](https://baike.baidu.com/item/%E5%AF%86%E7%A0%81%E6%95%A3%E5%88%97%E5%87%BD%E6%95%B0" \t "https://baike.baidu.com/item/SHA-1/_blank)，[美国国家安全局](https://baike.baidu.com/item/%E7%BE%8E%E5%9B%BD%E5%9B%BD%E5%AE%B6%E5%AE%89%E5%85%A8%E5%B1%80" \t "https://baike.baidu.com/item/SHA-1/_blank)设计，并由美国国家标准技术研究所（NIST）发布为联邦数据处理标准（FIPS）。SHA-1可以生成一个被称为消息摘要的160[位](https://baike.baidu.com/item/%E4%BD%8D" \t "https://baike.baidu.com/item/SHA-1/_blank)（20[字节](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82" \t "https://baike.baidu.com/item/SHA-1/_blank)）散列值，散列值通常的呈现形式为40个[十六进制](https://baike.baidu.com/item/%E5%8D%81%E5%85%AD%E8%BF%9B%E5%88%B6/4162457" \t "https://baike.baidu.com/item/SHA-1/_blank)数。  实现：  class SHA1 { //对message散列得160位，取5位（32,64,96,128,160）随机数，送入最终结果的文件。  public:  void creat\_w(unsigned char input[64], unsigned long w[80])  {  int i, j; unsigned long temp, temp1;  for (i = 0; i < 16; i++)  {  j = 4 \* i;  w[i] = ((long)input[j]) << 24 | ((long)input[1 + j]) << 16 | ((long)input[2 + j]) << 8 | ((long)input[3 + j]) << 0;  }  for (i = 16; i < 80; i++)  {  w[i] = w[i - 16] ^ w[i - 14] ^ w[i - 8] ^ w[i - 3];  temp = w[i] << 1;  temp1 = w[i] >> 31;  w[i] = temp | temp1;  }  }  char ms\_len(long a, char intput[64])  {  unsigned long temp3, p1; int i, j;  temp3 = 0;  p1 = ~(~temp3 << 8);  for (i = 0; i < 4; i++)  {  j = 8 \* i;  intput[63 - i] = (char)((a & (p1 << j)) >> j);  }  return '0';  }  void begin(unsigned long in)  {  unsigned long H0 = 0x67452301, H1 = 0xefcdab89, H2 = 0x98badcfe, H3 = 0x10325476, H4 = 0xc3d2e1f0;  unsigned long A, B, C, D, E, temp, temp1, temp2, temp3, k, f; int i, flag; unsigned long w[80];  unsigned char input[64]; long x; int n;  input[0] = in >> 24;  input[1] = in >> 16;  input[2] = in >> 8;  input[3] = in;  n = strlen((LPSTR)input);  if (n < 57)  {  x = n \* 8;  ms\_len(x, (char\*)input);  if (n == 56)  for (i = n; i < 60; i++)  input[i] = 0;  else  {  input[n] = 128;  for (i = n + 1; i < 60; i++)  input[i] = 0;  }  }  creat\_w(input, w);  A = H0; B = H1; C = H2; D = H3; E = H4;  for (i = 0; i < 80; i++)  {  flag = i / 20;  switch (flag)  {  case 0: k = 0x5a827999; f = (B & C) | (~B & D); break;  case 1: k = 0x6ed9eba1; f = B ^ C ^ D; break;  case 2: k = 0x8f1bbcdc; f = (B & C) | (B & D) | (C & D); break;  case 3: k = 0xca62c1d6; f = B ^ C ^ D; break;  }  temp1 = A << 5;  temp2 = A >> 27;  temp3 = temp1 | temp2;  temp = temp3 + f + E + w[i] + k;  E = D;  D = C;  temp1 = B << 30;  temp2 = B >> 2;  C = temp1 | temp2;  B = A;  A = temp;  }  H0 = H0 + A;  H1 = H1 + B;  H2 = H2 + C;  H3 = H3 + D;  H4 = H4 + E;  writ(H0);  writ(H1);  writ(H2);  writ(H3);  writ(H4);  }  void writ(unsigned long tp) {  int tmp = 0;  string a;  tmp = tp % 2;  if (tmp == 0)  {  a = "0"; cnt0++;  }  else  {  a = "1"; cnt1++;  }  rfile << a;  }  };   * 1. **得到最终1 000 0000个随机数。（10MB）**   重复上述过程200 0000次，5\*200 0000=1000 0000，得到最终的结果。   1. **NIST测试**   上诉过程得到1000 0000位2进制随机数，存储在txt文档，random.txt，共10MB。之后使用NIST随机数测试套件，进行检测。  **测试过程：**  1000 0000万个数据分为10个bitstream，每个大小100 0000.      **测试结果：**    Freq.txt：其中存储了10个bitstream中0,1的数量。    FinalAnalysisReport.txt。其中为15个测试总的测试结果。其中各项无’\*’号时说明通过该项测试。其中NonOverlappingTemplate较长，截取部分，完整见报告中FinalAnalysisReport.txt。P-VALUE为分析结果，一般P-VALUE大于0.01即为通过该项测试。此处P-VALUE为10个P-VALUE卡方均值。由文件可知测试文档random.txt通过NIST测试。            其余15个文件夹中存储各项分析结果。  以FFT为例。Results.txt为10个bitstream的P-value.stats.txt为10个bitstream的具体分析。Success即为通过测试。         1. **结果：**   综上所述，上述方法生成的随机数成功通过NIST测试，故上述随机数生成器为合格的真随机数生成器。 |
| 1. 实验过程分析   （实验分工，详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格等） |
| **遇到的问题及解决：**   1. **对于NIST的使用，及测试标准不清楚**   解决：查阅官网文档，明白了测试标准，查阅网络博客学会了测试过程。  **实验结果记录：**  随机数生成器代码为最终提交的文件夹中的random.cpp。  生成的1000 0000位2进制随机数存储在最终提交的文件夹中的random.txt。  NIST测试结果为最终提交的文件夹中的AlgorithmTesting文件夹。    综上所述，上述方法，利用汇编指令通过获取cpu硅片上的热噪声为seed，经过Xorshift，梅森旋转与SHA-1算法散列后，生成的随机数成功通过了NIST测试，故上述随机数生成器为合格的真随机数生成器。 |