

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **课程名：** | 操作系统 | **实验报告** |  | **实验日期：** | 2024.11.29 |
| **学院：** | 计算机与信息学院 | **网选班级：** | 5班 | **任课教师：** | 蔡政英 |
| **学号：** | 202210120510 | **姓名：** | 向申赤 | **评分：** |  |

|  |  |
| --- | --- |
| **实验名称**： | 实验5 同步与互斥 |
| **1. 实验目的：**   * 实现生产者消费者程序； * 生产者进程和消费者进程通过共享内存实现进程间通信交换数据。   **2. 实验环境及材料**：  **2.1 硬件环境配置：**  **CPU：**   * CPU架构: x86\_64 * CPU型号: Intel(R) Xeon(R) Gold 6133 CPU @ 2.50GHz * CPU核心数: 1   **内存：**   * CPU占用: 11.80% * 物理内存: 615.81/896.02 MB (68.73%) * 虚拟内存: 518MB/2047MB (25%)   **硬盘：**   * 硬盘占用: 13/28GB (49%)11/28GB (41%)   **2.2 软件名称、版本号及参数配置：**  **操作系统：**   * 系统版本: Debian GNU/Linux 12 (bookworm) * Linux版本: 6.1.0-27-amd64   **仿真软件：**  gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0  QTerminal 1.2.0  FeatherPad 1.3.5  **3. 实验内容**：  **3.1 实验设计（包括实验现场照片、电路图、真值表、主要代码的截图）**  按照实验要求分别创建以下C语言文件（producer.c、consumer.c、utils.h），并使用GCC编译器编译生成可执行文件后在终端中运行：        **3.2 实验实施（包括实验步骤、代码参数调整截图、不同工况下实验结果截图或照片）**  在Linux系统上准备实验环境。创建源代码文件（consumer.c, producer.c, utils.h）复制到实验环境目录中。  编译源代码。在终端中，导航到源代码所在目录，然后使用gcc命令编译consumer.c和producer.c文件。编译命令可能如下所示：  gcc -o consumer consumer.c  gcc -o producer producer.c  编译成功后，在两个不同的终端窗口中分别运行消费者和生产者程序。运行命令如下：  ./consumer  ./producer  仔细观察并记录两个程序的输出。关注生产和消费的过程，特别是缓冲区的状态变化和同步行为。这个观察过程是理解生产者-消费者问题核心的关键步骤。  为了深入理解程序的行为，尝试调整代码中的各种参数。在utils.h文件中修改BUFFER\_SIZE的值，或者在producer.c和consumer.c中调整生产和消费的速率。每次调整后，重新编译程序并运行实验，观察这些变化如何影响程序的行为。  ①在producer.c和consumer.c生产和消费的速率正常情况下，修改BUFFER\_SIZE的值，对比其运行情况。  BUFFER\_SIZE=1（极端情况）：正常运行      BUFFER\_SIZE=100（正常工况）：正常运行    BUFFER\_SIZE=10000（极端工况）：出现错误    ②在BUFFER\_SIZE=100（正常工况）下，调整producer.c和consumer.c生产和消费的速率，对比其调佣系统资源的情况。  未修改producer和consumer：    Producer生产一个产品后暂停1s：    Consumer消费一个产品后暂停1s：    Producer和Consumer均在生产和消费一个产品后暂停1s：    **3.3 实验数据分析（包括误差分析或实验失败原因分析）**：  在实验中，通过调整缓冲区大小和生产、消费速率，得出了多组数据和行为特征，这些实验数据揭示了系统运行的核心特点以及潜在问题。  首先，对于缓冲区大小(BUFFER\_SIZE)的调整，实验展现了极端条件和常规条件下的运行特点。**当BUFFER\_SIZE=1时**，缓冲区容量最小，生产者和消费者进程频繁等待对方的操作完成。这种情况下，虽然程序运行正常，但由于频繁的上下文切换和锁定开销，系统资源的消耗显著增加，尤其在生产和消费速率较高时表现尤为明显。**对于BUFFER\_SIZE=100的设置**，实验显示了系统的最佳运行状态，生产者和消费者能够高效交替运行，缓冲区始终保持适中的利用率，不会因资源不足或过量占用而导致性能问题。而**当缓冲区扩大到BUFFER\_SIZE=10000时**，系统出现了运行错误。这可能是由于共享内存分配超出了系统允许的范围，或者生产者和消费者在缓冲区操作上出现逻辑问题，进一步说明缓冲区大小的合理设置对系统的稳定性至关重要。  其次，调整生产和消费的速率进一步展示了系统在动态条件下的运行特性。**当生产者在每次生产后暂停1秒时**，消费者可能因数据不足而陷入等待，导致缓冲区利用率降低，但整体运行稳定。而**消费者在每次消费后暂停1秒时**，生产者的快速生产使缓冲区频繁接近满状态，这可能引发潜在的溢出风险。当**两者都设置为1秒的暂停时间时**，生产和消费达到了较好的平衡状态，缓冲区的利用率保持在一个相对稳定的中间水平，系统资源的使用也较为均衡。  最后，错误分析指出了程序在极端情况下的脆弱性。当缓冲区过大时，不仅可能超出系统的共享内存限制，还可能因数据溢出或指针计算超出范围而导致程序运行失败。此外，生产和消费速率不匹配时也可能导致缓冲区空耗或溢出，进一步说明合理配置各项参数对于并发程序的稳定性的重要性。  **4. 实验思考（包括实验心得体会）：**  通过本次实验，我深刻认识到了生产者消费者问题中的同步与互斥机制，以及其在实际应用中的挑战与意义。实验表明，合理设计共享资源的容量和管理策略是实现并发程序高效运行的关键。缓冲区的大小直接影响生产者和消费者的交互效率，过小的缓冲区会导致频繁的等待和资源浪费，而过大的缓冲区则可能引发系统错误和资源占用过多的问题。这个结论在实际开发中具有重要的指导意义，例如在设计数据库连接池或任务队列时，可以参考类似的分析方法。  此外，实验中通过调整生产和消费速率，验证了动态条件下系统的适应性和平衡能力。在实际应用中，生产者和消费者的速率往往会因外部条件而变化，因此可以设计更智能的动态调整机制。例如，根据缓冲区的当前状态，实时调整生产或消费的频率，以保证系统的平稳运行和资源的高效利用。  实验还揭示了共享内存和进程通信中的潜在问题，例如内存分配限制和进程间同步的实现细节。为此，在程序设计时需要考虑全面的错误处理机制，例如检测系统允许的共享内存上限，避免指针越界操作，并为可能的异常提供清晰的恢复路径。此外，引入信号量或条件变量替代简单的轮询等待机制，可以显著降低CPU资源的占用，提高程序运行的效率。  综上，本次实验通过调整关键参数和分析系统行为，全面深入地理解了生产者消费者问题在并发编程中的实际应用价值。这种理论结合实践的学习方法，不仅帮助掌握了同步与互斥的基本原理，也提升了我面对复杂问题时分析和解决的能力。未来，在设计更复杂的系统时，可以参考类似的实验方法，优化资源分配和任务调度策略，从而实现高效、可靠的系统运行。 | |