

Introduction

Having multiple entities that all run simultaneously sharing data in some form or other, requires you to either design some scheme where protection is needless *OR* employ some form of *protection* and/or *signaling*. In this lecture we will look closely at the possible protection and signaling mechanisms available. This entails their concept and use.

Content and reflection

Themes

- Protection and Synchronization[3, chap. 30][1]
- Posix semaphores[3, chap. 53][1]
excluding named
- Buffer case - Producer/Consumer[2][1]
Implemented using semaphores

Questions

- Shared data
 - What is shared data
 - What can and *will* go haywire with multiple threads - why
 - What to do about it
- Protection
 - Which different protection mechanisms are possible
 - What characterised each of these
 - How does the API look like and how do you use it
- Signalling
 - Which different signalling mechanisms are possible
 - What characterised each of these
 - How does the API look like and how do you use it
 - The notation *spurious wakeup* applies to *conditional variables* - how and what does this mean
- The Car park
 - How does it work conceptually
 - Determine who waits on who and when
- Producer/Consumer
 - Concept
 - From an implementation point of view, how does it work

Material

Slides

- [1] S. Hansen, *Thread synchronization i*, Slides - see course repos.

Online

- [2] E. al. (). “Producer–consumer problem.” Wikipedia Article, [Online]. Available: https://en.wikipedia.org/wiki/Producer%2dconsumer_problem.

Hardback

- [3] M. Kerrisk, *The Linux Programming Interface*. No Starch Press, Inc, 2010, ISBN: 978-1-59327-220-3.