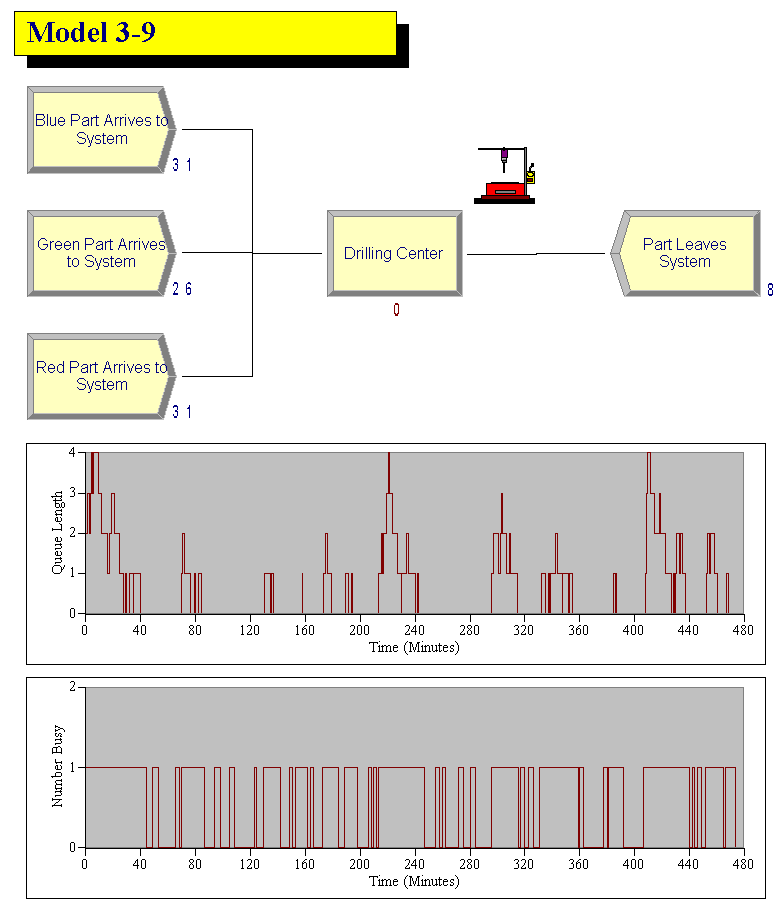
**<Ch.3 – 9>**

문제 상황에 맞게, 우선 Blue/Green/Red Ball Entity를 만들어준 뒤 Time between arrivals를 mean이 16인 지수 분포로 설정한다. 이를 각각 Create Process로 설정하고, 시스템이 480분 동안 작동하므로 Run의 setup에서 Replication Length를 480으로 설정한 뒤, 두 Plot의 x축 역시 maximum을 480으로 바꾸어 주면, 아래와 같은 모델을 만들 수 있다.



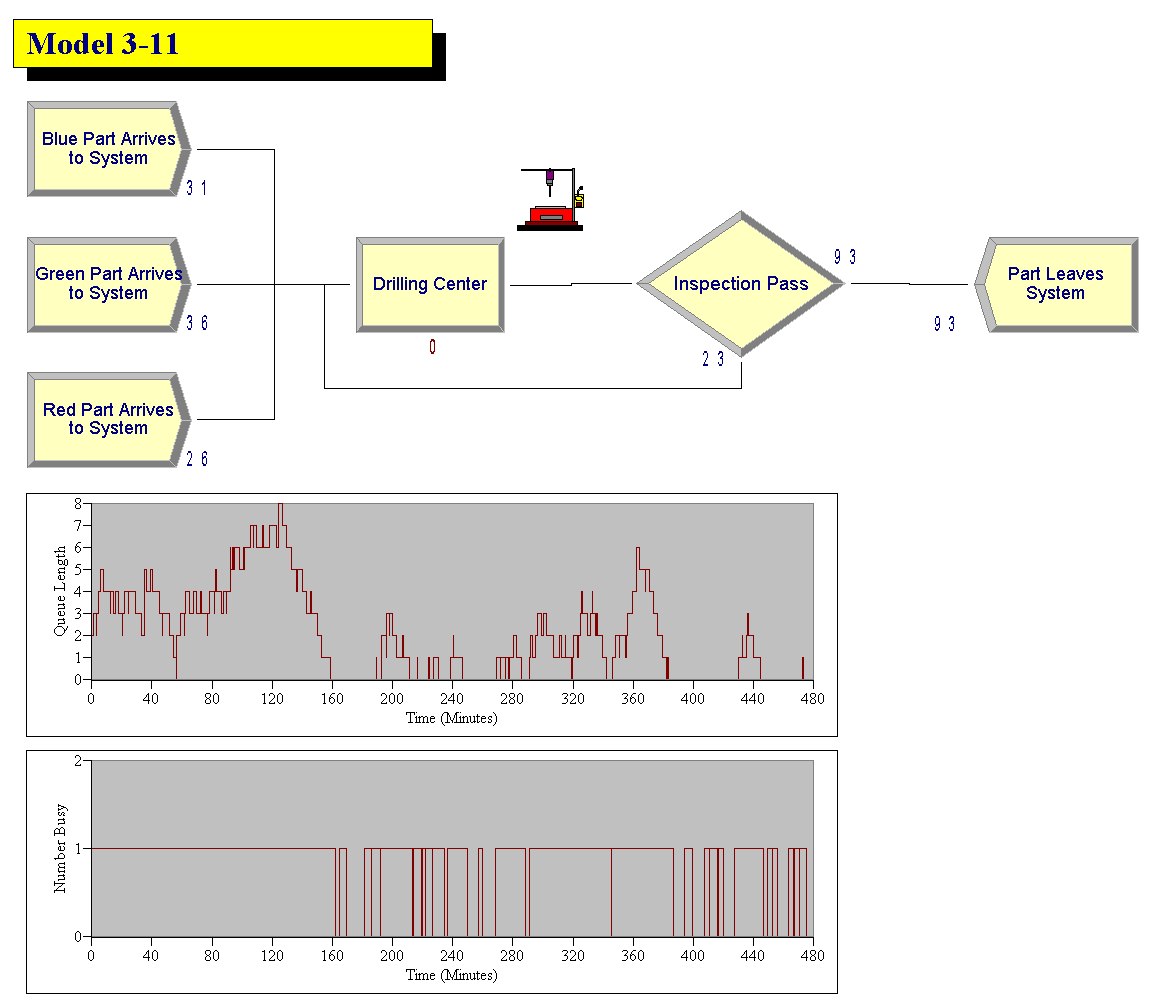
이를 실행하여 performance measures를 살펴보면,

|  |  |  |  |
| --- | --- | --- | --- |
|  | Average | Minimum | Maximum |
| time in queue for all part types together | 3.0077 minutes | 0 minutes | 15.6421 minutes |
| queue length for all part types together | 0.5514 parts | 0 parts | 4 parts |
| server utilization for all part types together | 0.6474 | 0 | 1 |
| Average time-in-system results |  |  |  |
| -Blue Part | 6.8337 minutes | 2.011 minutes | 19.0057 minutes |
| -Green Part | 7.1205 minutes | 2.2155 minutes | 16.8567 minutes |
| -Red Part | 5.7569 minutes | 1.9194 minutes | 14.2642 minutes |

Replication을 1개만 만들도록 했으므로 Half Width는 파악할 수 없다.

**<Ch.3 – 11>**

3-9 모델에, Decide 모듈을 추가하여 85% 확률로 검사를 통과하고, 15% 확률로 part를 재공정받게 하면, 아래와 같은 모델을 만들 수 있다.



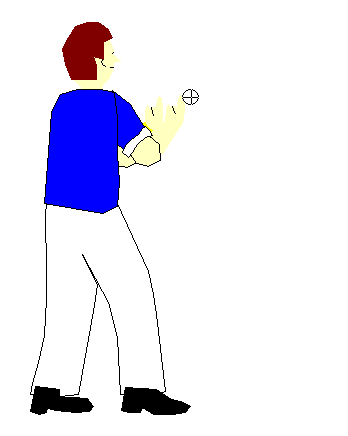
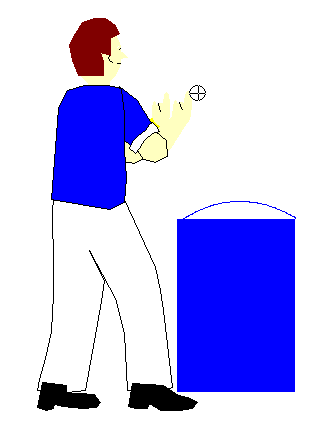
전과 동일하게 performance measures를 살펴보아 비교해보면,

|  |  |  |
| --- | --- | --- |
|  | 3-9 | 3-11 |
| Avg time in queue for all part types together | 3.0077 minutes | 8.1488 minutes |
| Avg queue length for all part types together | 0.5514 parts | 1.9693 parts |
| Avg server utilization for all part types together | 0.6474 | 0.8420 |
| Avg time-in-system results |  |  |
| -Blue Part | 6.8337 minutes | 13.6994 minutes |
| -Green Part | 7.1205 minutes | 15.3285 minutes |
| -Red Part | 5.7569 minutes | 14.3421 minutes |

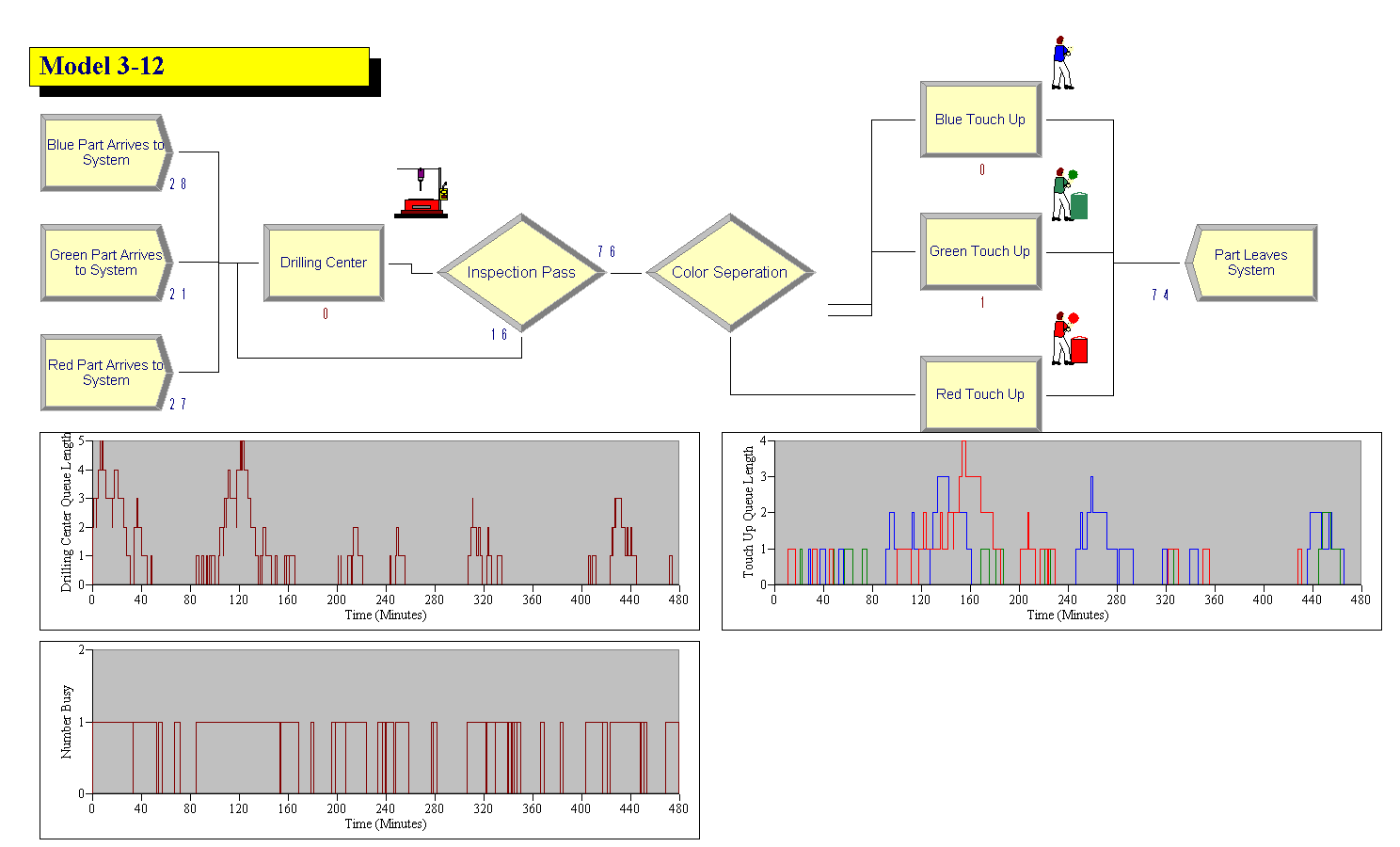
Queue에서의 시간, Queue의 평균 길이, 가용률, 각 Part의 총 걸린 시간들의 평균 모두 3-9와 비교하여 크게 증가했다. 15% 확률로 재공정을 받게 했기에 이러한 결과가 명백한 사실처럼 보이지만, 단언할 수는 없다. 이는 단 한 번의 Replication일 뿐이므로, 이 결과로부터는 어떠한 사실도 이끌어낼 수 없기 때문이다. 여러 번의 Replication을 반복하여 Half Width와 같은 통계량을 구해 둘을 비교한다면, 두 상황이 다름을 그때에서야 주장할 수 있을 것이다.

**<Ch.3 – 12>**

Touch-up을 진행하기 위해, 우선 재공정까지 끝난 각 parts를 색깔에 맞게 분류한다. Decide 모듈의 N-way by condition을 이용한다. 그 후, 각 색깔 별 Process와 Resource를 만들어준다. 문제 상황에 맞게 각 Resource는 1unit씩 seize/release하고, 각 Process는 TRIA(3, 10, 18) minutes를 따르도록 설정한다. Drill Press와 같이 각 Resource에 대한 그림도 그려주면, 아래와 같다.

:idle, : busy

마지막으로, Touch Up Queue의 길이도 Plot으로 만들어주면, 아래와 같은 모델을 만들 수 있다.



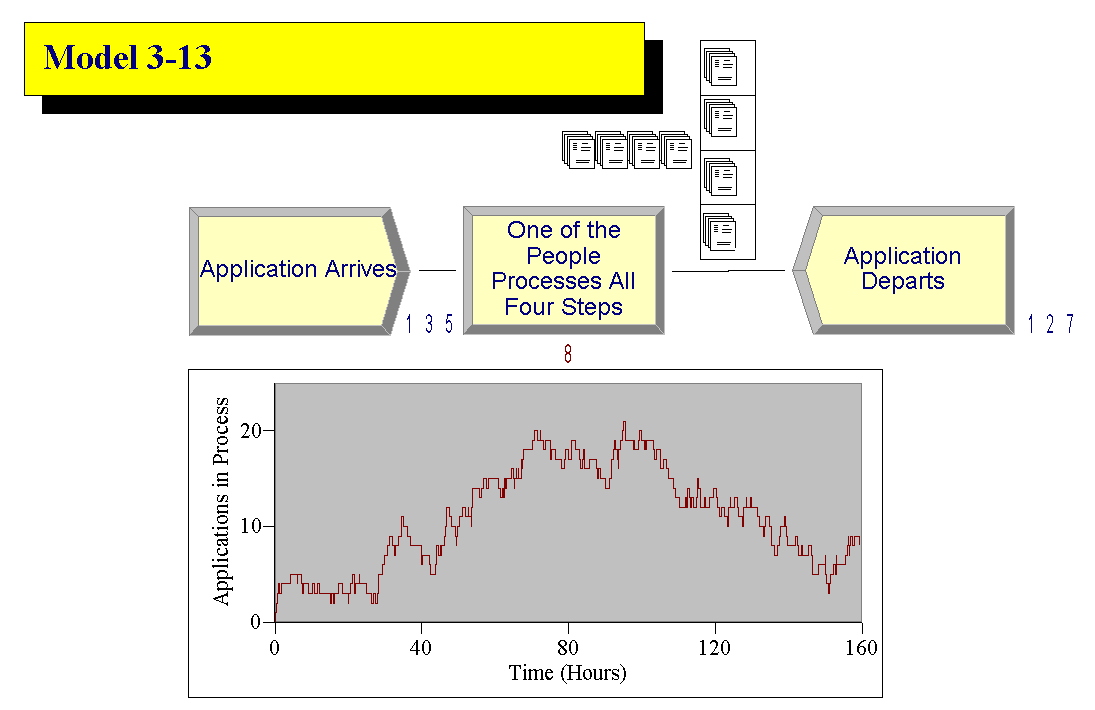
전과 동일하게 performance measures를 살펴보아 비교해보면

|  |  |  |  |
| --- | --- | --- | --- |
|  | Average | Minimum | Maximum |
| Avg time in queue for all part types together |  |  |  |
| -Drilling Center | 3.6298 | 0 | 17.0563 |
| -Blue Touch Up | 8.5544 | 0 | 28.3203 |
| -Green Touch Up | 2.6798 | 0 | 14.1167 |
| -Red Touch Up | 7.1924 | 0 | 31.9524 |
| Avg queue length for all part types together |  |  |  |
| -Drilling Center | 0.6957 | 0 | 5 |
| -Blue Touch Up | 0.499 | 0 | 3 |
| -Green Touch Up | 0.1172 | 0 | 2 |
| -Red Touch Up | 0.4046 | 0 | 4 |
| Server utilization for all part types together |  |  |  |
| -Drilling Center | 0.6389 | 0 | 1 |
| -Blue Touch Up | 0.6554 | 0 | 1 |
| -Green Touch Up | 0.4675 | 0 | 1 |
| -Red Touch Up | 0.5814 | 0 | 1 |
|  | | | |
| Average time-in-system results |  |  |  |
| - Blue Part | 28.5085 | 11.7216 | 47.7529 |
| - Green Part | 21.9088 | 7.7754 | 35.2347 |
| - Red Part | 26.6055 | 6.4447 | 56.4441 |

마찬가지로, 이는 1 번의 Replication 결과일 뿐이다.

**<Ch.3 – 13>**

이 모델처럼 분업하지 않고 작업을 통합해서 한다면, 각 작업을 완료하는 데에 걸리는 시간이 증가하는 것은 당연하다. 연구를 통해 1.18배 증가한다는 것이 문제에서 주어졌으므로, 3-3 모델에서 프로세스 부분을 기존 EXPO( 1 ) + EXPO( 1 ) + EXPO( 1 ) + EXPO( 1 )에서, 1.18 \* (EXPO( 1 ) + EXPO( 1 ) + EXPO( 1 ) + EXPO( 1 ))으로 바꾸어주면, 원하는 모델을 만들 수 있다.



이 모델의 결과를 통해 과연 이러한 일반화된 병렬 통합 작업 방식이 분업을 적용한 직렬 작업 방식보다 여전히 효과적인지 생각해보면,

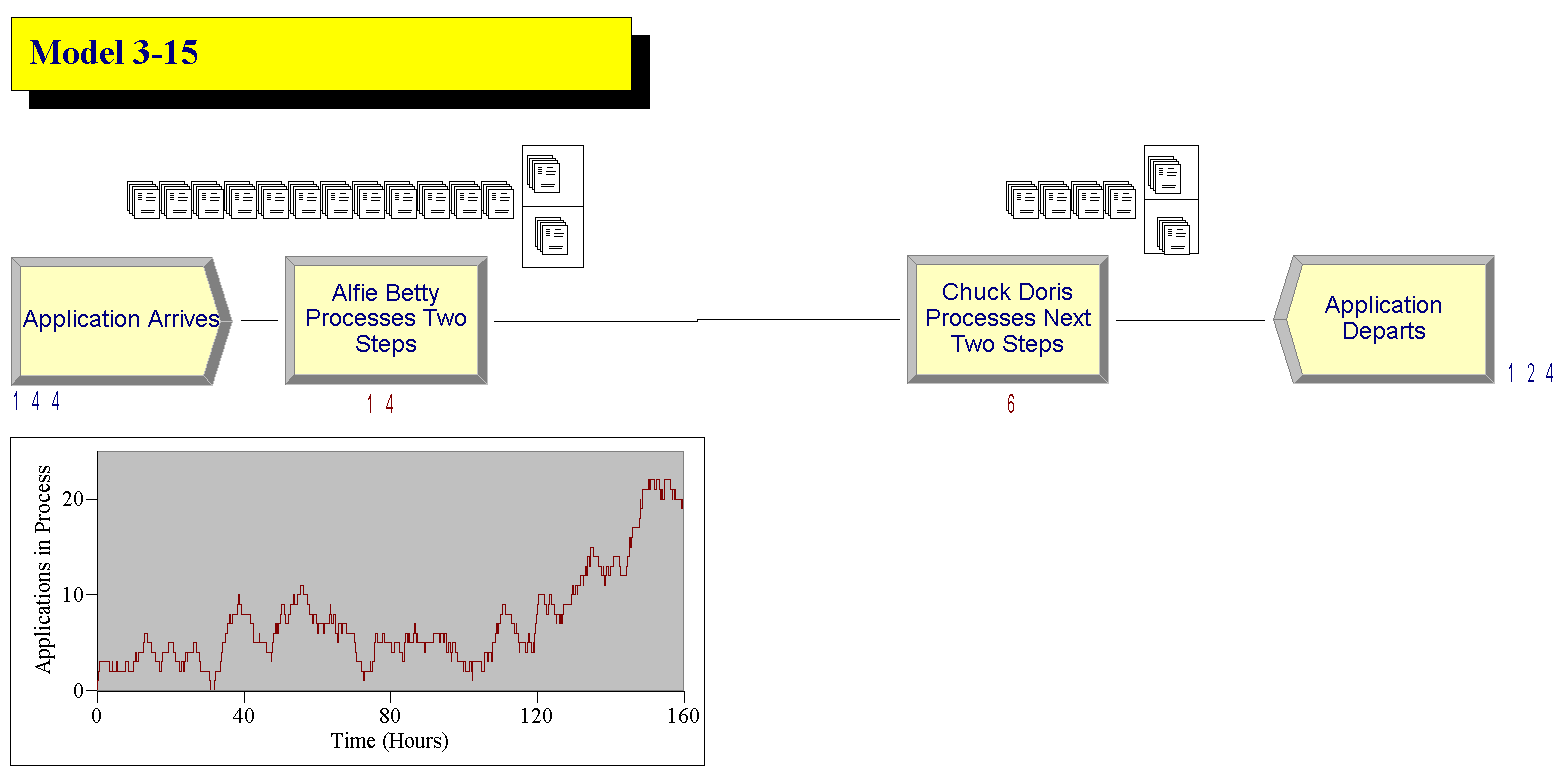
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 모델 | Total WIP | | Total Time  In System | | Total  Waiting Time | | Number Processed | Avg.  Utilization |
| Avg | Max | Avg | Max | Avg | Max |  |  |
| 3-2  (직렬) | 12.39 | 21 | 16.08 | 27.21 | 11.98 | 22.27 | 117 | 0.78 |
| 3-3  (병렬) | 4.61 | 10 | 5.38 | 13.73 | 1.33 | 6.82 | 135 | 0.87 |
| 3-13  (일반화 병렬) | 10.86 | 21 | 13.37 | 25.71 | 8.60 | 20.07 | 127 | 0.97 |

여전히 병렬적인 통합 처리(3-13)가 직렬 분업 처리(3-2)보다 생산량, 가용률, 대기시간, 총 시간 면에서 더 생산적인 결과를 얻을 수 있었지만, 작업 시간의 18% 증가를 고려하기 전 병렬 통합 처리(3-3)에서 보여졌던 큰 차이가 많이 좁혀진 것을 알 수 있다.

하지만 이 역시 한 번의 시행일 뿐이므로, 어떠한 결론도 내릴 수는 없겠다.

**<Ch.3 – 15>**

4명의 직원에게 4개의 작업을 모두 교육시키는 것이 아닌, 직원 2명에게 첫 2 작업을 교육시키고, 다른 2명에게 남은 2 작업을 교육시키는 식으로 3-2와 3-3 모델을 절충한 상황이다. 이를 모델에 적용하기 위해, Resource를 2개 만든 후, 2개의 Process는 2개의 작업씩 하므로 EXPO( 1 ) + EXPO( 1 ) 식을 이용해 만든다. Resource 그림까지 그려주면, 아래의 모델을 만들 수 있다.



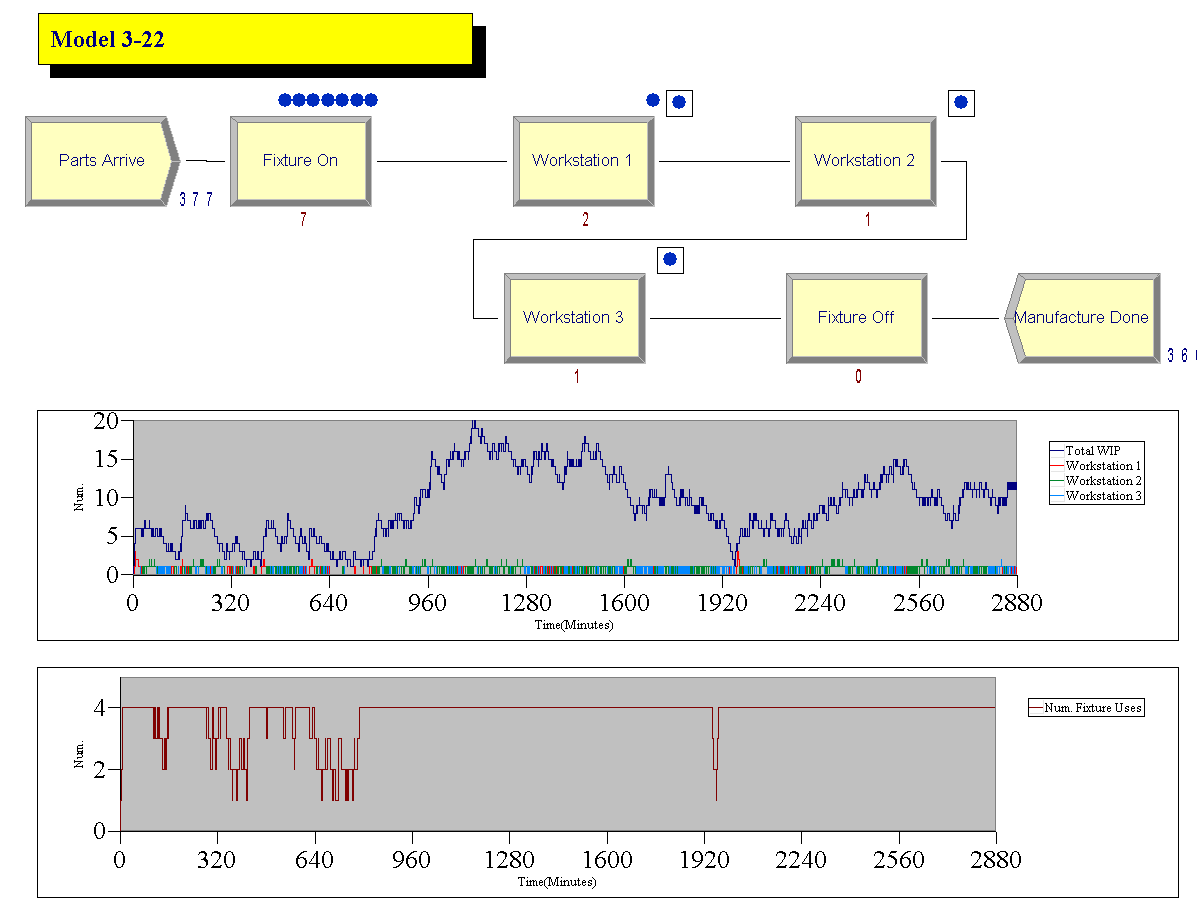
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 모델 | Total WIP | | Total Time  In System | | Total  Waiting Time | | Number Processed | Avg.  Utilization |
| Avg | Max | Avg | Max | Avg | Max |  |  |
| 3-2  (직렬) | 12.39 | 21 | 16.08 | 27.21 | 11.98 | 22.27 | 117 | 0.78 |
| 3-3  (병렬) | 4.61 | 10 | 5.38 | 13.73 | 1.33 | 6.82 | 135 | 0.87 |
| 3-15  (2직렬  +2병렬) | 7.36 | 22 | 7.86 | 19.98 | 3.71 | 16.8 | 124 | (0.86+0.79)/2  =0.83 |

이와 같은 결과를 얻을 수 있다. 우리는 3-15에서와 같이 직렬과 병렬적 연결을 혼합한 형태가 가장 효율적일 것으로 기대한다. 하지만, 위 표의 결과는 절충안이 전체 병렬 연결보다 덜 생산적인 것처럼 보이게 한다.

이는 이번에도 역시 단 한 번의 결과일 뿐이므로, 우리의 사전지식과 동일한 결론을 내리기 위해서는 여러 번의 Replication을 반복할 필요가 있다.

**<Ch.3 – 15>**

문제 상황에 맞게, Resource를 만든 후, Create, Process, Dispose를 적절히 배치하면 아래와 같이 원하는 모델을 만들 수 있다. 세부사항은 첨부파일을 참조하길 바란다.

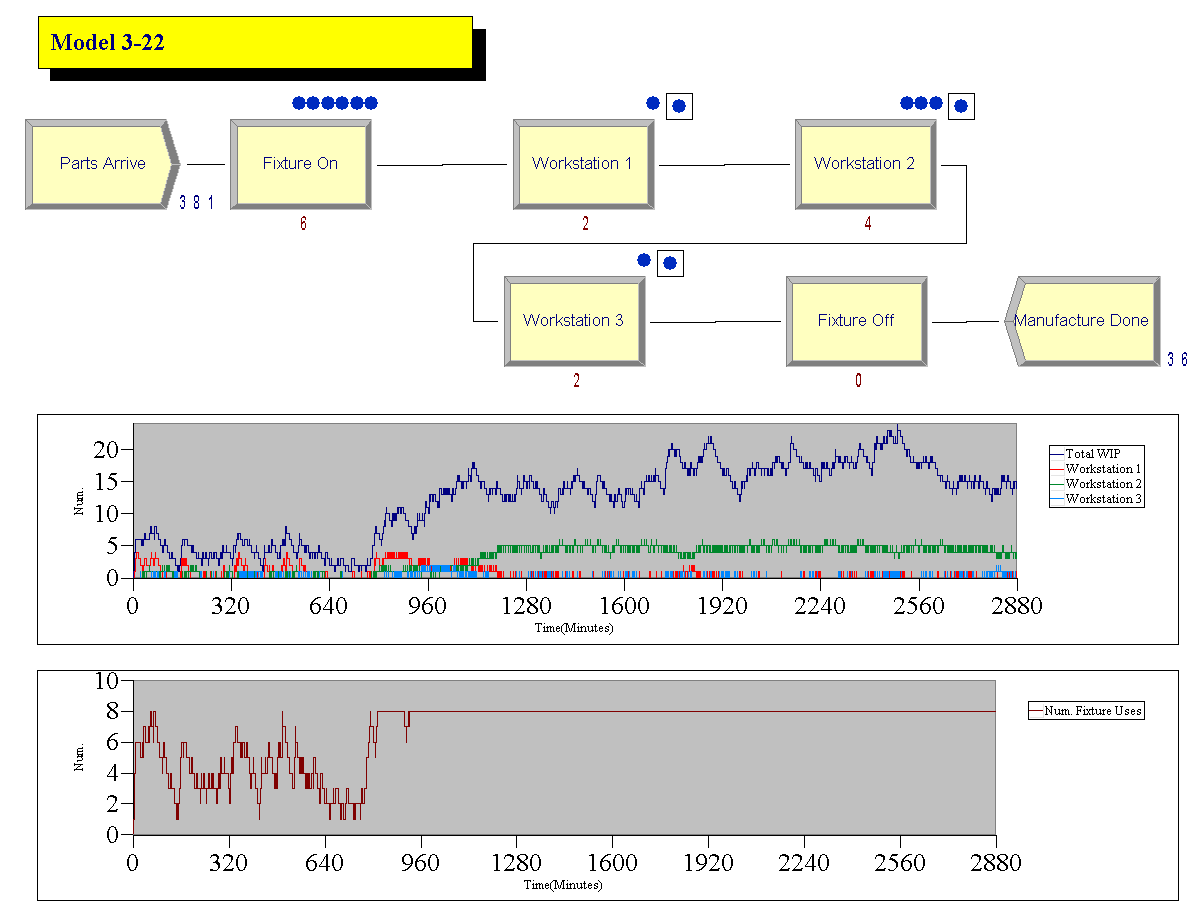


Single Replication을 요구했으므로 1개의 Replication으로 실행한 모델의 결과는,

|  |  |  |
| --- | --- | --- |
|  | Average | Maximum |
| Total work in process (WIP) | 8.95 | 20 |
| Waiting Time |  |  |
| -Workstation 1 | 1.09 | 16.36 |
| -Workstation 2 | 3.73 | 12.46 |
| -Workstation 3 | 1.04 | 8.87 |
| -Fixtures | 39.65 | 119.82 |
| Queue Length |  |  |
| -Workstation 1 | 0.14 | 3 |
| -Workstation 2 | 0.48 | 2 |
| -Workstation 3 | 0.13 | 2 |
| -Fixtures | 5.15 | 16 |
| Resource Utilization |  |  |
| -Workstation 1 | 0.86 | 1 |
| -Workstation 2 | 0.94 | 1 |
| -Workstation 3 | 0.87 | 1 |
| -Fixtures | 0.95 | 1 |

1번의 시행 결과일 뿐이지만, 결과가 타당하다는 가정 하에 이를 분석해보겠다.

Waiting Time과 Queue Length, 그리고 Resource Utilization으로 미루어보아, 이 시스템의 효율성을 낮추는 병목현상이 일어나는 부분은 Fixture 부분이다. 처리해야 하는 3개의 공정을 4개씩만 Fixture한 후에 진행할 수 있으므로, 고정 장치를 4개보다 많은 수로 늘리면 효율성이 향상될 것으로 기대된다. 실제로 Fixture Resource의 Capacity를 4에서 8로 수정하고 실행해보았다. 그 결과로 나온 그래프와 각 지표들의 평균값을 비교한 표는 아래와 같다.



|  |  |  |
| --- | --- | --- |
|  | 기존 상황 | Fixture 4 -> 8 |
| Avg. Total work in process (WIP) | 8.95 | 12.24 |
| Avg. Waiting Time |  |  |
| -Workstation 1 | 1.09 | 4.54 |
| -Workstation 2 | 3.73 | 23.61 |
| -Workstation 3 | 1.04 | 1.46 |
| -Fixtures | 39.65 | 40.65 |
| Avg. Queue Length |  |  |
| -Workstation 1 | 0.14 | 0.59 |
| -Workstation 2 | 0.48 | 3.05 |
| -Workstation 3 | 0.13 | 0.19 |
| -Fixtures | 5.15 | 5.34 |
| Avg. Resource Utilization |  |  |
| -Workstation 1 | 0.86 | 0.86 |
| -Workstation 2 | 0.94 | 0.96 |
| -Workstation 3 | 0.87 | 0.87 |
| -Fixtures | 0.95 | 0.86 |

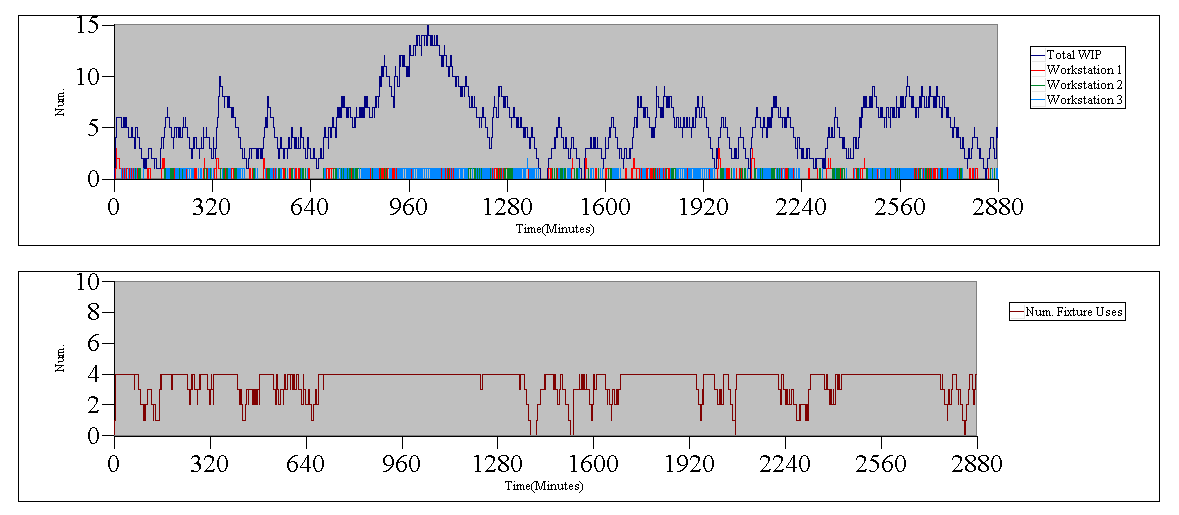
기대한 대로 효율성이 향상되지 않고, 오히려 감소한 것을 확인할 수 있다. 이를 검증하기 위해 Fixture Resource의 Capacity을 4에서 8로 늘린 상황을 20개의 Replication으로 확인해보면,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 기존 상황 | Half Width | Fixture 4 -> 8 | Half Width |
| Avg. Total work in process (WIP) | 9.1 | 1.8 | 8.8 | 1.84 |
| Avg. Waiting Time |  |  |  |  |
| -Workstation 1 | 1.17 | 0.16 | 6.29 | 0.96 |
| -Workstation 2 | 4.02 | 0.27 | 16.61 | 3.48 |
| -Workstation 3 | 0.78 | 0.07 | 1 | 0.18 |
| -Fixtures | 42.53 | 13.21 | 21.32 | 10.74 |
| Avg. Queue Length |  |  |  |  |
| -Workstation 1 | 0.14 | 0.02 | 0.78 | 0.11 |
| -Workstation 2 | 0.5 | 0.04 | 2.1 | 0.48 |
| -Workstation 3 | 0.1 | 0.01 | 0.12 | 0.02 |
| -Fixtures | 5.42 | 1.72 | 2.83 | 1.45 |
| Avg. Resource Utilization |  |  |  |  |
| -Workstation 1 | 0.82 | 0.02 | 0.83 | 0.02 |
| -Workstation 2 | 0.92 | 0.02 | 0.82 | 0.02 |
| -Workstation 3 | 0.83 | 0.02 | 0.84 | 0.02 |
| -Fixtures | 0.92 | 0.02 | 0.75 | 0.06 |

Avg. WIP과 Fixtures의 Avg. Waiting Time, Avg. Queue Length을 신뢰구간을 이용해 비교해보았을 때, 유의미하게 차이가 난다고 결론내릴 수 없다. 대신, Workstation 2에서의 차이가 두드러지는데, 이로부터 이 System의 병목현상은 Workstaion2로부터 발생하고, 이 때문에 고정 장치에서도 병목 현상이 일어난다고 생각해볼 수 있다.

이를 확인하기 위해, Fixture Resource의 Capacity를 다시 8에서 4로 복원한 후, Workstation2의 성능을 끌어올려 최대 10분 걸리던 공정을 8분으로 줄인 상황을 생각했다. 즉, Workstation2가 U(5,10)이 아닌 U(5,8)을 따르도록 수정한 뒤 마찬가지로 20개의 Replication을 생성했다..

Replication 1의 그래프와 20번의 Replication 결과를 비교한 표는 아래와 같다.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 기존 상황 | Half Width | Workstation2 ~ U(5,10) ->U(5,8) | Half Width |
| Avg. Total work in process (WIP) | 9.1 | 1.8 | 6.42 | 0.85 |
| Avg. Waiting Time |  |  |  |  |
| -Workstation 1 | 1.17 | 0.16 | 2.11 | 0.11 |
| -Workstation 2 | 4.02 | 0.27 | 0.9 | 0.06 |
| -Workstation 3 | 0.78 | 0.07 | 1.65 | 0.1 |
| -Fixtures | 42.53 | 13.21 | 22.66 | 6.08 |
| Avg. Queue Length |  |  |  |  |
| -Workstation 1 | 0.14 | 0.02 | 0.27 | 0.01 |
| -Workstation 2 | 0.5 | 0.04 | 0.11 | 0.01 |
| -Workstation 3 | 0.1 | 0.01 | 0.21 | 0.02 |
| -Fixtures | 5.42 | 1.72 | 2.93 | 0.81 |
| Avg. Resource Utilization |  |  |  |  |
| -Workstation 1 | 0.82 | 0.02 | 0.84 | 0.02 |
| -Workstation 2 | 0.92 | 0.02 | 0.82 | 0.02 |
| -Workstation 3 | 0.83 | 0.02 | 0.86 | 0.02 |
| -Fixtures | 0.92 | 0.02 | 0.87 | 0.02 |

Avg. WIP과 Fixtures의 Avg. Waiting Time, Avg. Queue Length을 신뢰구간을 이용해 비교해보았을 때, 유의미하게 차이가 난다고 결론내릴 수 있다. 이로부터 이 System의 병목현상은 Workstaion2로부터 발생한다고 주장할 수 있다.